Notebook

April 13, 2019

1 Confidence Intervals and Bayesian Statistics oh my!

One of the readings for week 13, "The Bayesian New Statistics," covers a variety of different approaches to statistics, as contrasted with the standard frequentist hypothesis-testing method. I don't expect you to come out of this class being able to work any of those alternative paradigms, but you should be able to recognize them and understand broadly how they operate. That article is a very good summary of the landscape, but this supplemental lesson aims to provide a briefer and slightly more basic introduction.

1.1 I. Confidence Intervals vs P-Values

One of the ideas in that article is "new statistics," which is roughly meant to capture the move away from p-values and hypothesis tests and toward confidence intervals. This movement has arisen in recent years mainly because of the many ways in which hypothesis tests and p-values have been misused. The idea is that reporting a confidence interval is a clearer expression of the range of uncertainty of the thing you're trying to estimate.

So what's a confidence interval? Roughly speaking, it's the range equivalent of a p-value—a lower bound and an upper bound for a parameter constructed to achieve a specified degree of confidence (typically 95%).

Interpreting confidence intervals can be difficult, however. The standard interpretation of them is "if we took a bunch of samples from the population, then calculated a 95% confidence interval from each of the samples, we'd expect to see the population (true) parameter within 95% of the confidence intervals in our hypothetical creation of many samples. Here's a concise definition from some epidemologists and here's an explanation of common misinterpretations of confidence intervals as well as p-values.

Statsmodels reports confidence intervals for regression coefficients. Let's take a look at them, and then I'm going to show you one (fairly brutish—there are more sophisticated and accurate ways to do it in the real world) way to create your own via bootstrapping.

First, let's simulate some data. We're going to pretend that heights are normally distributed around 6 feet with a wide standard deviation (they're not), and then we're going to pretend that shoe sizes are about height in inches/10, with some Gaussian (normally distributed) noise added to represent, say, observation error, or other factors leading to shoe size other than height (diet? amount of soccer played as a child?).

```
In [1]: import numpy as np
        import pandas as pd
        import statsmodels.formula.api as sm
```

```
        np.random.seed(90210)
```

```
In [2]: heights = np.random.normal(loc=72.0, scale=5.0, size=500)
        shoe_sizes = [(height / 10) + np.random.normal(scale=2.0) for height in heights]
        heights = heights.astype(int)
        shoe_sizes = np.array(shoe_sizes).astype(int)
```

So we have a simulated sample of heights and shoe sizes, of size 500, and we made them integers (which I believe should just truncate the decimals, though I could be misremembering how Numpy does it, and it might round—probably truncates though) to add some more noise. Now let's wrap them up in a Data Frame and look at a regression coefficient.

```
In [3]: df = pd.DataFrame({"height": heights, "shoe": shoe_sizes})
```

```
        reg = sm.ols(formula='shoe ~ height', data=df).fit()
        print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   shoe   R-squared:                       0.070
Model:                            OLS   Adj. R-squared:                  0.068
Method:                 Least Squares   F-statistic:                     37.39
Date:                Sat, 13 Apr 2019   Prob (F-statistic):           1.95e-09
Time:                        14:09:45   Log-Likelihood:                -1050.2
No. Observations:                 500   AIC:                             2104.
Df Residuals:                     498   BIC:                             2113.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -1.1321      1.270     -0.891      0.373      -3.628       1.363
height         0.1089      0.018      6.115      0.000       0.074       0.144
==============================================================================
Omnibus:                        6.090   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.048   Jarque-Bera (JB):                6.220
Skew:                           0.268   Prob(JB):                       0.0446
Kurtosis:                       2.894   Cond. No.                     1.02e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.02e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Some things to note here (ignoring the silliness about the condition number, which is related to troubles that statsmodels has had with this calculation).

1. Our coefficient looks about right—pretty close to .1, which is what we would expect given that how the data were simulated.

2. In the right-most two columns of the middle section of the table, we see confidence intervals—in this case, a 95% confidence interval goes from 0.074 to 0.144.

Now let's bootstrap a confidence interval! This can be computationally intensive, and I'm impatient, so in order to do this, I'm going to *try* to make it faster by calculating the regression coefficients directly rather than messing around with library code that calculates a bunch of other stuff, and puts into a DataFrame (or manipulates statsmodels in other ways to avoid going through slow Pandas) etc. etc. This might be a terrible mistake, though, as hand-written code by me is almost always slower than library code. Still, let's see what happens.

Mathematically, we can calculate a simple (one independent variable) linear regression coefficient with the following equation, where $\bar{x}$ is the mean of x and subscripts index the individual points in the dataset:

$$\beta = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

This is pretty trivial to reduce to code.

```
In [4]: def beta(pairs):
            x = [point[0] for point in pairs]
            mean_x = np.mean(x)
            y = [point[1] for point in pairs]
            mean_y = np.mean(y)
            numerator = 0
            for point in pairs:
                numerator += (point[0] - mean_x) * (point[1] - mean_y)
            denominator = np.sum([np.square(item - mean_x) for item in x])
            return numerator / denominator
```

My function takes a single array of (x, y) points, representing a single data point, which we can get using Python's built-in `zip()` function. Let's make sure I did the math right by comparing the results of this function to the built-in statsmodels function.

```
In [5]: pairs = np.array(list(zip(heights, shoe_sizes)))

        coefficient = beta(pairs)
        print(coefficient)
```

```
0.10885151617958266
```

Looks good to me! It's the same, minus some rounding difference. Now we can do some bootstrapping. We're going to randomly resample, with replacement (so they're different) from our existing sample to generate a lot of synthetic datasets, and then we're going to generate a 95% confidence interval by taking the 2.5%ile and the 97.5%ile of the betas calculted from each.

```
In [6]: def resample(pairs):
            size = len(pairs)
            indices = np.random.choice(a=size, size=size, replace=True)
            sample = pairs[indices]
            return beta(sample)
        bootstrap_samples = np.array([resample(pairs) for x in range(10000)])
        print(np.percentile(bootstrap_samples, 2.5))
        print(np.percentile(bootstrap_samples, 97.5))
```

```
0.07409887356837877
0.14371195712135248
```

Note how we got a result very close to what statsmodels calculated for us (again, basically indistinguishable, modulo rounding)!

So, that's confidence intervals. When the author you're reading this week talks about the "new statistics," essentially what they mean is "reporting confidence intervals rather than reporting the result of a hypothesis test," with the idea being that you're effectively estimating the true value of your parameter and then expressing a range in which you're reasonably confident that it lies, rather than a simple binary "effect/no effect." If you must, you can translate back and forth between the paradigms by saying that if the confidence interval includes the null hypothesis value of the parameter (often 0) then you won't have a significant result.

Recommended further reading: this American Psychological Association essay on estimation vs hypothesis testing.

Now let's talk about Bayesian statistics.

1.2 II. Bayesian Statistics

Recall that all frequentist statistics are fundamentally based on the idea of the sampling distribution. We have certain beliefs (derived from things like the central limit theorem) about what kinds of data the world will show us, under given assumptions, and then we look at the data that the world has, in fact, shown us, and ask how much does our data embarass the assumptions about the world that we started with.

By contrast, the foundational question of Bayesian statistics is "given some prior belief about the world, and some evidence, what do we believe now?" I don't want to dig too deeply into this, as we don't even remotely have the time to cover it. But the basic idea is:

1. Come up with a prior probability distribution. For example, in a discrimination case, you might ask "what's the probability that men and women are paid differently?" This can be an "informative prior"—one that's based on, for example, prior research. Or it can be an "uninformative prior" that more-or-less doesn't bias the results in one way or another.

2. Collect data.

3. Apply Bayes rule. The prior distribution gives us the probability of seeing our data, and we can then update quite directly to get a posterior distribution.

There's a good clear explanation of the basic ideas of bayesian vs frequentist inference in this old MIT lecture notes document. The major issue that lots of people have with Bayesian inference

is the business with priors. The choice of an uninformative or an informative prior can matter a lot, and which is appropriate depends a lot on context. For example, a Bayesian statistician trying to learn about whether telekenesis exists would probably be well-advised to use an informative prior, because a freak result in an experiment probably shouldn't budge your beliefs very much. A statistician trying to figure out whether a particular employer engaged in discrimination probably ought to start with an uninformative prior, because it would unfairly put a thumb on the scale to begin with a belief that the employer discriminates against women, say, to the same degree as the observed nationwide pay gap.

In the past, the major barriers to the widespread adoption of Bayesian statistics have probably been the scarier math and the computational demands; these days computers are lots faster so it's the scarier math and the priors. (The math does get quite scary when you get to complicated questions... although honestly the math in frequentist maximum likelihood estimation, which I haven't even talked about in this class, also gets a little scary-looking and calculus-ey.) But the flip side to the priors issue is the overwhelming advantage of Bayesian statistics, namely that it answers the question we *actually want to answer,* viz, how likely is the hypothesis given our data, rather than how likely is the data given the hypothesis.

A second advantage is Bayesian statistics is that collecting more data isn't cheating. In the frequentist null hypothesis significance testing approach, collecting data, looking at it, seeing a nonsignificant p-value, and then going out and getting more data, is totally p-hacking; you're looking at the data a bunch of times, and so you're radically increasing the risk of type I errors (of falling into that 5% of cases where you get a significant result when the null hypothesis is true). But you don't have to worry about stopping rules in Bayesian hypothesis testing, because more data unproblematically gets integrated into the overall estimate. (This is a *slight* oversimplification... Bayesian statistics folks have endless debates about stopping rules. But if there are problems, they aren't nearly as severe as the problem of "peeking" at the data in the middle of data collection in a frequentist/null-hypothesis context.)

This class has focused on classical statistics mainly because I believe that this is what you are most likely to encounter in legal practice.