**Overview – Classical Item Analysis**

Today we will:

* Calculate & interpret item difficulty and discrimination parameters.
  + Point-biserial correlations.
  + Biserial correlations.
  + Remainder score correlations.
  + Interval estimation.

**Item Difficulty**

Open the two Excel files labeled “Lab 11 Example 1” and “Lab 11 Example 2.”

* Example 1 gives data for 6 items on a test of math attitudes (scored on a Likert scale from 1 to 5).
* Example 2 gives data for 6 items on a test of math knowledge (scored as 0 = incorrect or 1 = correct).

The item difficulty parameter for **both** binary and continuous data is the mean. You have several tools available to calculate item difficulty parameters

* Using the =AVERAGE function in Excel.
* Using describe() (from the ‘psych’ package) in R
* Using ANALYSIS: TYPE = BASIC in Mplus.

Choose your favorite method, and see if you can do it mostly from memory.

* In each test, which item is the most difficult?
* Based on the item difficulty parameters, at what level(s) of the construct should each test measure best?

Easy, right? Yes and no.

Most of the time, we are satisfied with estimating item difficulty parameters for a population based on the sample statistics. But if we have a small sample, or if we need precise information about difficulty, it can be advantageous to look not just at the **point estimate** but also at the **confidence interval** around the item difficulty estimates.

**Item Discrimination Parameters**

Essentially, the item discrimination parameter is the correlation between item score and total test score. However, we have a number of choices for how we want to calculate that correlation. For continuously scored items, we have:

* The ordinary Pearson correlation

And for binary items, we have:

* The point-biserial correlation
* The biserial correlation

We *can* calculate these in Mplus, but it is tedious and inefficient, and we could get point-biserials in Excel, but we can’t easily get the biserial. In R, though, we can get them both. Read in the Example 1 and Example 2 datasets:

data.con <- read.csv(file.choose(), header=TRUE)

data.bi <- read.csv(file.choose(), header=TRUE)

Choose the Example 1 file for data.con and the Example 2 file for data.bi.

To get any kind of item-total correlation, the first step is to calculate a total. So we’ll add a variable to each data frame that is the sum of the item scores:

data.con$total <- apply(data.con, 1, sum)

data.bi$total <- apply(data.bi, 1, sum)

I’m using the apply() function to tell R to sum across all of the other values in each row. This saves having to type all of the item names. I can get the ordinary Pearson correlations (which, remember, are the point-biserial correlations for binary data) by using the cor() function on each data set:

cor(data.con)

cor(data.bi)

Because our data set includes the total score, the item-total correlations are in the last row and column. For our continuous data, we’re done.

# Biserial Correlations

For the binary data, though, we can do better. Recall that the point-biserial correlation **systematically underestimates** the true correlation when we believe that the binary item is “falsely” dichotomizing an underlying construct that is really continuous. For example, your answer to an algebra item is scored as correct or incorrect, but your underlying knowledge of algebra is continuous. You might know just enough to pass this item, or you might know a lot more than that. The biserial correlation takes this into account by allowing us to treat that dichotomous item as an indicator of an underlying continuous variable.

We can get the biserial correlations by using the biserial() function in the psych package:

biserial(data.bi$total, data.bi[,1:6])

The first argument to the biserial() function is the continuous variable (“total”). The second argument selects columns 1-6 of our overall data frame (i.e., just the columns with the raw item data in them). This tells R to calculate the biserial correlations between the total and each item, so we get a total of 6 biserial correlations.

Let’s compare the results from the point-biserial and biserial correlations:

Point-biserial Biserial

V1 0.7019253 0.8930654

V2 0.7486607 0.9874583

V3 0.3566087 0.8895682

V4 0.6644721 0.8428219

V5 0.7026632 0.9454380

V6 0.6891270 0.8740943

Pretty big differences! We would come to different conclusions about some of these items if we relied on the point-biserial. Note in particular the difference for item, number 3. The point-biserial yields a pretty low correlation – if you look back at the item difficulty (.97!), you’ll notice that this is a *really easy* item. There’s not much variance in the item response, so the range restriction limits the correlation. The biserial allows us to take into account that this is still a pretty diagnostic item (if you *didn’t* pass this item, that tells us a lot!).

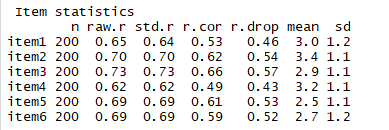
# Remainder Score Correlations

As we discussed in lecture, though, all of these correlations are somewhat inflated, because the item we are focusing on is included in the total. This means that part of our overall correlation is due to the perfect relationship between the item and itself. If we have a large number of items, the impact of this overlap is minor, but in a smaller item set (like 6 items!) it can make a difference. To control for this, we can calculate our item-total correlations based on the **remainder score**: the total score on all of the items **except** the one we’re correlating.

An easy way to obtain these remainder score correlations for continuous items is to use the alpha() function. We’ve used this one before; load the ‘psych’ package, then run:

alpha(data.con)

At the end of your output, you will get a table of descriptive statistics:



In this table, the values in the raw.r column are the basic item-total correlations. The r.drop values are the item-remainder score correlations. (The std.r column = the item-total correlation if the items are standardized, and the r.cor column = the item-total column corrected for unreliability, both of which are only occasionally useful). In this case, you can see that the correlations are quite a bit lower when we use the remainder scores; this gap diminishes as we increase the number of items, but with only 6 items it’s quite noticeable.

**Remainder Score Biserials**

You can use the alpha() function to get *point­*-biserial correlations for the binary items as well, as these are just the ordinary Pearson correlation. However, if you want biserials for the remainder scores, you have to do a little more work – just a little.

This is a good time to develop your R programming skills just a tiny bit and learn how to use a programming technique called a **for loop**. We’re going to tell R to go through a series of steps **for** each item – this way, we only have to tell it once.

First, a little setup. We create a variable to tell R how many items we have (how many times it needs to run the loop); this allows us to make our loop a little more general, should we want to use it later on. We then set up a matrix called “remainder.corrs” to hold the values we need. To make life easy, we’ll put the point-biserial and biserial correlations in the same matrix (point-biserials in column 1 and biserials in column 2).

nitems <- 6

remainder.corrs <- matrix(rep(0, nitems\*2), ncol=2)

Then, we need to calculate the total scores. The apply() function is great for creating sums and averages. The first argument (“data.bi”) is the data you want to aggregate. The next argument (“1”) tells R to aggregate over **rows** (if you entered “2” here, you’d get the **column** totals instead). Finally, the last argument is the function you want to use; find “sum” rather than “mean” here, because that’s going to make it easier to get the *remainder* scores.

totals <- apply(data.bi, 1, sum)

Now, we start the for loop. This tells R to repeat these steps for every value *i* in the sequence from 1 to nitems (that is, 1-6). As R runs the loop, the first time through, it will insert “1” every time it sees “*i*". The next time through, it will insert “2”, and so on.

for (i in 1:nitems) {

remainder.corrs[i,1] <- cor((totals-data.bi[,i]), data.bi[,i])

remainder.corrs[i,2] <- biserial((totals-data.bi[,i]), data.bi[,i])

}

To get the remainder scores, instead of figuring out which 5 items to add each time and finding the sum, we find the sum of all 6 items first and then just subtract the item we’re interested in.

The steps inside the loop are:

* Calculate the correlation between (the total score – the item score) and the item score. Store this value in row i, column 1 of remainder.corrs.
* Calculate the biserial correlation between (the total score – the item score) and the item score. Store this value in row i, column 2 of remainder.corrs.

We use a bracket ( } ) to close the loop, then tell R to print our completed remainder.corrs matrix. When we run this code, we get:

> remainder.corrs

[,1] [,2]

[1,] 0.5067772 0.6447769

[2,] 0.5884311 0.7761209

[3,] 0.2647903 0.6605251

[4,] 0.4530032 0.5745930

[5,] 0.5305542 0.7138642

[6,] 0.4873084 0.6181060

These values are *substantially* less than the values based on the full total scores. However, the biserial correlations are still larger than the point-biserials.

**Lab Exercise:**

Use the “Lab 11 Data” Excel file from Canvas.

1. Calculate and report your item difficulty parameters. Which items are easiest? Which are most difficult?
2. Use R to calculate point-biserial and biserial correlations between **each** item and:
   1. The total score including the item
   2. The remainder score
3. Report all four sets of correlations.
4. For this data, which set of item discrimination parameters is most appropriate? Why?
   1. Do any of the calculations seem problematic to you? Explain.
5. Imagine that you have been asked to reduce this test from 10 items to 5, but still measure across a broad range of the construct. Which 5 items do you keep and why?
   1. Support your answer based on the data above. Be specific and explain your rationale for **each** item.

Turn in (a) a Word doc with your answers to the above and (b) your R script, or an R Markdown document integrating both.