01_statistical_inference - Week 4

Notes on the course
https://www.coursera.org/learn/statistical-inference

Andres FR

August 2020

Reading: Power

We've talked about a Type I error, rejecting the null hypothesis when it's true. We've structured our hypothesis test so that the probability of this happening is small. The other kind of error we could make is to fail to reject when the alternative is true (Type II error). Or we might think about the probability of rejecting the null when it is false. This is called Power = 1 - Type II error. We don't have as much control over this probability, since we've spent all of our flexibility guaranteeing that the Type I error rate is small.

One avenue for the control of power is at the design phase. There, assuming our finances let us, we can pick a large enough sample size so that we'd be likely to reject if the alternative is true. Thus the most frequent use of power is to help us design studies.

Power:

> **Power** is the probability of rejecting H_0 when it is false. The probability of a Type II error (failing to reject H_0 when false) is usually called β, and the power is then $1 - \beta$.

- It's a good thing, you want more power.

In a low-power experiment (e.g. very few samples), failing to reject H_0 is expected simply due to noise: we just won't be able to reject, and this will be a Type II error if it's false.

For this reason, **power is a more relevant indicator for null results than for non-null results**, and **is a relevant factor when designing as study**, since we don't want to wait for a null outcome to realize it had low power.

Examples: The power will depend on the obtained statistic, but also on the H_0 mean. Consider this: If we expected 30 (our μ_a), but obtained 60, the power would be very big. On the other hand, if we expected 30, but we obtained 30.01, the power would be much smaller.

$$P\left(\frac{\bar{X}-30}{s/\sqrt{n}} > t_{1-\alpha,n-1} \ ; \ \mu = \mu_a\right)$$

Calculating Power

Gaussian data Let's assume our statistic is normally distributed. We reject if:

$$\frac{\bar{X}-30}{\frac{\sigma}{\sqrt{n}}} > z_{1-\alpha}$$

And we have:

$$H_0 : \bar{X} \sim N(\mu_0, \frac{\sigma^2}{n}) H_a : \bar{X} \sim N(\mu_a, \frac{\sigma^2}{n})$$

To calculate it with r, see the following snippet:

```
alpha <- 0.05
mu_0 = 30
mu_a = 31
sigma <- 1
n <- 10
pnorm(mu_0 + qnorm(1 - alpha)*sigma/sqrt(n), mean=mu_a,
      sd=sigma/sqrt(n), lower.tail=FALSE)
```

```
## [1] 0.9354202
```

An intuitive explanation for this can be the following:

1. Given a distribution for H_0 and α, we set the threshold for rejection. All statistics above it will be rejected, and this will generate type I errors with probability α.
2. Now take a different distribution for H_a. The same rejection threshold from before will also divide the domain here, but the areas to the left and right will be different: The area to the left corresponds to β, the probability of a Type II error: The cases when H_a is right but we failed to reject H_0. The area to the right is $1-\beta$, **the power** of the experiment.

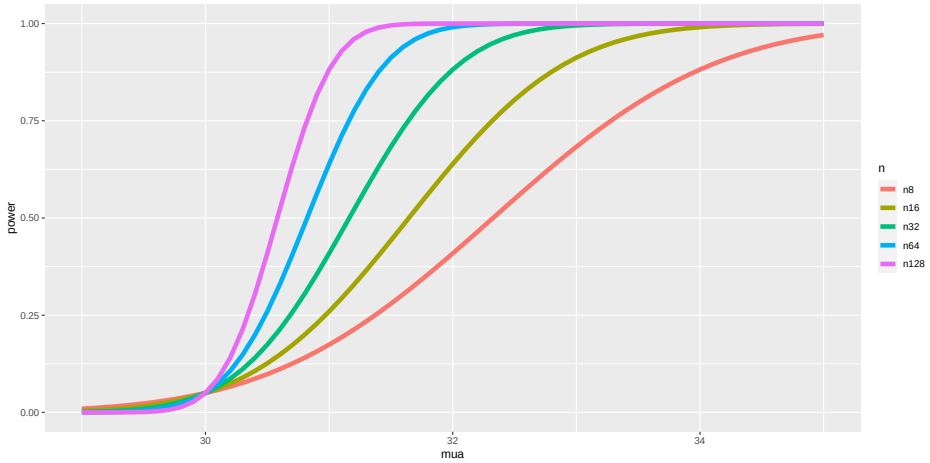
3. When we have a lot of samples, the H_0 distribution will be narrow and therefore the threshold will be "low". The H_a distribution will be narrow as well, so it is likely that a lot of the H_a area is above the threshold and therefore there is a lot of power. But as the number of samples decrease, the variance for H_0 increases, and the threshold raises. Not only that, but the variance for H_a also increases, so there will be more density in the tails. These 2 effects combined cause the H_a area above threshold to decrease, and thus the power.

- Note how if we used `mean=mu_0` instead of `mu_a`, the last expression would yield `alpha`.

The following interactive snippet helps to give an intuition:

```
library(manipulate)
mu0 = 30
myplot <- function(sigma, mua, n, alpha){
    g = ggplot(data.frame(mu = c(27, 36)), aes(x = mu))
    g = g + stat_function(fun=dnorm, geom = "line",
                          args = list(mean = mu0,
                                      sd = sigma / sqrt(n)),
                          size = 2, col = "red")
    g = g + stat_function(fun=dnorm, geom = "line",
                          args = list(mean = mua,
                                      sd = sigma / sqrt(n)),
                          size = 2, col = "blue")
    xitc = mu0 + qnorm(1 - alpha) * sigma / sqrt(n)
    g = g + geom_vline(xintercept=xitc, size = 3)
    g
}
manipulate(
    myplot(sigma, mua, n, alpha),
    sigma = slider(1, 10, step = 1, initial = 4),
    mua = slider(30, 35, step = 1, initial = 32),
    n = slider(1, 50, step = 1, initial = 16),
    alpha = slider(0.01, 0.1, step = 0.01, initial = 0.05)
    )
```

The following plot shows the increase of power in an all-gaussian scenario as a function of μ_a. Observe how all lines converge at 0.05 when $\mu_0 = \mu_a$:



Notes on Power

When calculating power:

- Unknowns: μ_a, σ, n, β
- Knowns: μ_0, α

For any 3 unknowns, we can get the remainder.

> In fact, power depends only on the so-called **effect size**: difference between means divided by standard error:

$$\frac{\sqrt{n} \cdot (\mu_a - \mu_0)}{\sigma}$$

* The effect size is a 1-dimensional function and is unit-free. It gives a lot of information about the test power and can be interpretable across settings.

- For 2-sided, a good approximation is usually to do the 1-sided power for the $\alpha/2$ in the direction of μ_a and ignore the other tail, which is usually residual (e.g. in gaussian-like distributions)

> The power of a 1-sided test is greater than the associated 2-sided test

---

T-test power

Revisiting our definition, the power for a t-test is:

$$P(\frac{\bar{X} - \mu_0}{S/\sqrt{n}} > t_{1-\alpha,n-1}; \mu = \mu_a)$$

So we are computing our t-statistic as usual, and we will reject if the t-threshold is surpassed. But for the power we want to measure the probability **assuming that the H_a distribution is true**.

> But, unlike before, this will follow a different distribution: the **non-central t-distribution** (see wiki), a generalization of the t-distribution whenever the assumed mean differs from the measured one.

Here, it will suffice to know that `power.t.test` in R does this very well: Omit one of the arguments and it solves for it. These examples show different omissions; some of them have the same *effect size* (e.g. 2 in the first 3 examples) and return the same power for different parameters:

```
power.t.test(n = 16, delta = 0.5, sd=1,
             type = "one.sample",  alt = "one.sided")$power
```

```
## [1] 0.6040329
```

```
power.t.test(n = 16, delta = 2, sd=4,
             type = "one.sample",  alt = "one.sided")$power
```

```
## [1] 0.6040329
```

```
power.t.test(n = 16, delta = 100, sd=200,
             type = "one.sample", alt = "one.sided")$power
```

```
## [1] 0.6040329
```

```
power.t.test(power = .8, delta = 0.5, sd=1,
             type = "one.sample",  alt = "one.sided")$n
```

```
## [1] 26.13751
```

```
power.t.test(power = .8, delta = 2, sd=4,
             type = "one.sample",  alt = "one.sided")$n
```

```
## [1] 26.13751
```

```r
power.t.test(power = .8, delta = 100, sd=200,
             type = "one.sample", alt = "one.sided")$n
```

```
## [1] 26.13751
```

Note the following:

- `alt="one.sided"` means that the alternate hypothesis test is one-sided (e.g. greater-than).
- "one.sample" means a one-sample T-Test (basically the scenario covered so far).
- Usually we round up the required n.

Also, note that calculations for power can be quite involved and lead to a false sense of security. Even if this gives a more conservative estimate, a `power.t.test` is easy to understand and it is often a good practice to start with it.

Multiple Comparisons (Jeffrey Leek)

Usually, HT is overused: Computing many p-values from data, and reporting just the smallest one, or many small ones, can lead to false discoveries.

> When you perform more than one hypothesis test you have to do some sort of correction to make sure that you're not fooling yourself. This lecture is a little bit about how to do these corrections.

Two key components: error measure, and correction

Prof. Bradley Efron has a good compilation of materials on statistics: http://statweb.stanford.edu/~ckirby/brad/papers/

Also: Introduction to multiple testing (link), and Leek's own research.

Note that multiple testing is an entire subfield in statistics. A basic Bonferroni/BH correction is usually enough, BY under strong dependence.

History The classical period (Pearson, Fisher, Neyman, Hotelling, etc) dealt mainly with extracting most information out of a scientific experiment, usually not much data and in the form *is A better than B?*.

In the era of scientific mass production, we can produce much bigger datasets, but we are also charged with much more questions that have to be answered together, and errors can pile up if not accounted for. Which variables matter most among the thousands measured? How do you relate unrelated information?

Example: You have a dataset with a big population, with 100 possible subdivisions. If you test for a specific relation between a sub-population and a disease, with $\alpha = 0.05$, you can expect that 5 out of the 100 tests will come out as a rejection of H_0 even if H_0 is always true.

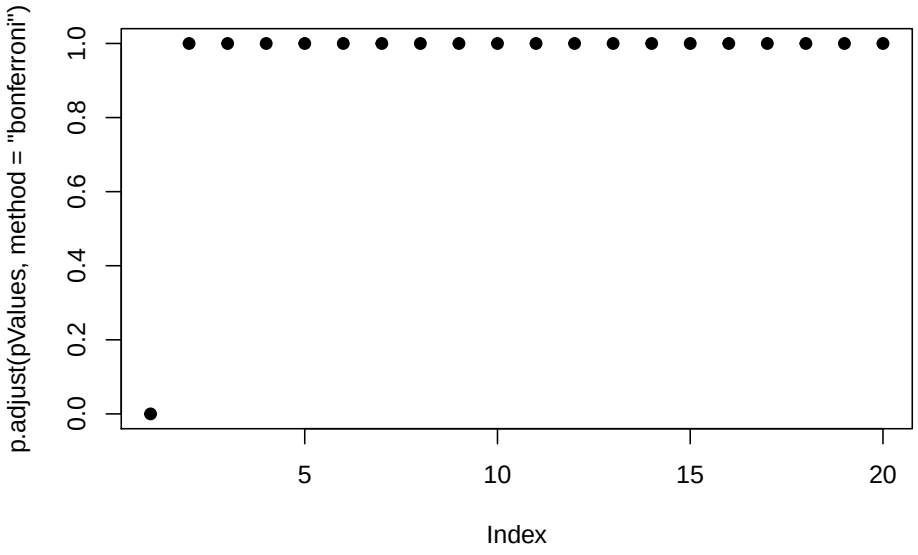
Types of errors:

- **False positive rate**: Rate for Type I errors among all instances where H_0 is true, i.e. instances where H_0 was wronfgully handled.
- **Family-wise error rate** (FWER): The probability of at least one false positive, $P(V \geq 1)$
- **False Discovery Rate** (FDR): The rate of type I errors among all rejections, i.e. instances where a rejection was wrong (independently of whether H_0 was true or not).

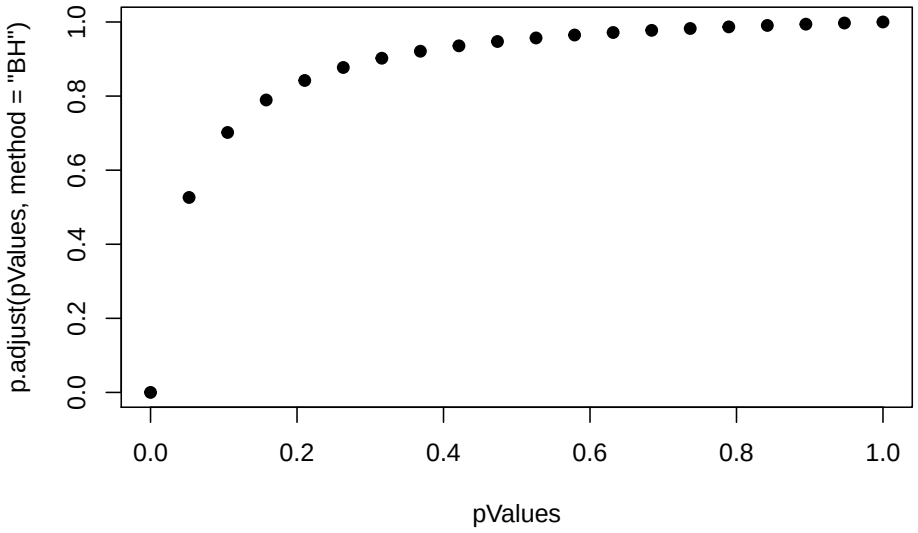
Corrections: Assuming we perform m tests,

- Bonferroni: the oldest multiple testing correction (FWER). We enforce $P(V \geq 1) = \alpha$, and for that we calculate the p-values normally and then simply divide: α/m.
  - Very conservative, but easy to calculate.
- Benjamini-Hochberg (BH): If we want to control FDR at a level α, we calculate the p-values normally, order them from smallest to largest and then call any $P_i \leq \frac{\alpha \cdot i}{m}$ significant
  - Much less conservative, still easy. But **may behave strangely under dependence**.
- Benjamini-Yekutieli (BY): Controls the FDR under arbitrary dependence assumptions: https://en.wikipedia.org/wiki/False_discovery_rate
- Adjusted p-values: Instead of adjusting α. This has good support in R, but once adjusted, they loose their semantics. Still they can be directly used with our original α.
  - Check code for examples

```
alpha <- 0.05
pValues <- seq(from = 0, to = 1, length.out=20)
plot(p.adjust(pValues, method="bonferroni"), pch=19)
```




```
plot(pValues, p.adjust(pValues, method="BH"), pch=19)
```



Case Study I: uncorrelated data In this example with uncorrelated data, note how the uncorrected thresholding returns about 5% of false positives, whereas the adjusted ones are much closer to zero:

```
set.seed(1010093)
pValues <- rep(NA,1000)
for(i in 1:1000){
  y <- rnorm(20)
  x <- rnorm(20)
  pValues[i] <- summary(lm(y ~ x))$coeff[2,4]
}
```

```r
# Controls false positive rate
sum(pValues < 0.05)
```

```
## [1] 51
```

```r
# Controls FWER
sum(p.adjust(pValues,method="bonferroni") < 0.05)
```

```
## [1] 0
```

```r
# Controls FDR
sum(p.adjust(pValues,method="BH") < 0.05)
```

```
## [1] 0
```

Case Study II: 50% correlated samples In this case, observe how the uncorrected thresholding still returns around 5% of false positives. The bonferroni correction will handle H_0 much more accurately, but fail to discover many H_a cases. The BH adjustment will have almost zero false negatives, but still some false positives.

```r
set.seed(1010093)
pValues <- rep(NA,1000)
for(i in 1:1000){
  x <- rnorm(20)
  # First 500 beta=0, last 500 beta=2
  if(i <= 500){y <- rnorm(20)}else{ y <- rnorm(20,mean=2*x)}
  pValues[i] <- summary(lm(y ~ x))$coeff[2,4]
}
trueStatus <- rep(c("zero","not zero"),each=500)
table(pValues < 0.05, trueStatus)
```

```
##        trueStatus
##         not zero zero
##   FALSE        0  476
##   TRUE       500   24
```

```r
# Controls FWER
table(p.adjust(pValues,method="bonferroni") < 0.05,trueStatus)
```

```
##        trueStatus
##         not zero zero
##   FALSE       23  500
##   TRUE       477    0
```

```r
# Controls FDR
table(p.adjust(pValues,method="BH") < 0.05,trueStatus)
```

```
##         trueStatus
##          not zero zero
##   FALSE        0  487
##   TRUE       500   13
```

Bootstrapping

- Very important tool for constructing confidence intervals and calculating standard errors, and basically performing inferences in difficult scenarios without need of heavy mathematics.
- Invented by Bradley Efron in 1979.
- E.g. *How would one derive a confidence interval for the median?* Mathematically complex, BS can provide that.
- It has a pretty clear working principle, in the spirit of data science.

Example scenario: If we have a fair roll, we know that the population distribution is uniform from 1 to 6.

If we want to know the distribution for the average of 50 rolls, there are multiple ways: We can do the algebra, we can use the CLT, or we can do simulations by drawing samples from the population distribution.

But what if we just have one sample (i.e. 50 draws) and don't know if the die is fair? We don't know the population distribution, so we can't sample from it, and doing math may be unfeasible.

In this scenario we can do bootstrapping: We make a histogram with the 50 samples as a proxy for the population distribution, and draw from there.

Bootstrapping Example

Consider the son height data in the father-son dataset. We can resample from that data many times (draw with replacement where each sample in the dataset has probability $\frac{1}{n}$) and compute the median each time as follows:

```
library(UsingR)
data(father.son)
x <- father.son$sheight
n <- length(x)
B <- 10000
resamples <- matrix(sample(x,
                           n * B,
                           replace = TRUE),
                    B, n)
resampledMedians <- apply(resamples, 1, median)
```

Notes on the Bootstrap

Procedure:

1. We have data drawn from an unknown pop dist
2. Draw from it with replacement to calculate a given statistic many times
3. Make a histogram for that statistic
4. Use the histogram to obtain confidence intervals, stddev, etc, from the statistic

Drawing from the data is approximately equal to draw from the pop dist (to the extent that the data approximates the distribution itself).

Nonparametric bootstrap algorithm example Given a dataset, we want a confidence interval for the median out of n samples.

1. Sample n with replacement
2. Extract the statistic (in this case the median)
3. Repeat steps 1 and 2 for B times (the number of **bootstrap resamples**). Since this is a monte carlo simulation, we want B to be large to minimize the associated approximation error.
4. These medians are approximately drawn from the sampling distribution. "There is a lot of theory that shows that BS actually works". You can then draw histograms, compute stddev, quantiles...

```
B <- 10000
resamples <- matrix(sample(x,
                           n * B,
                           replace = TRUE),
                    B, n)
medians <- apply(resamples, 1, median)
sd(medians)
```

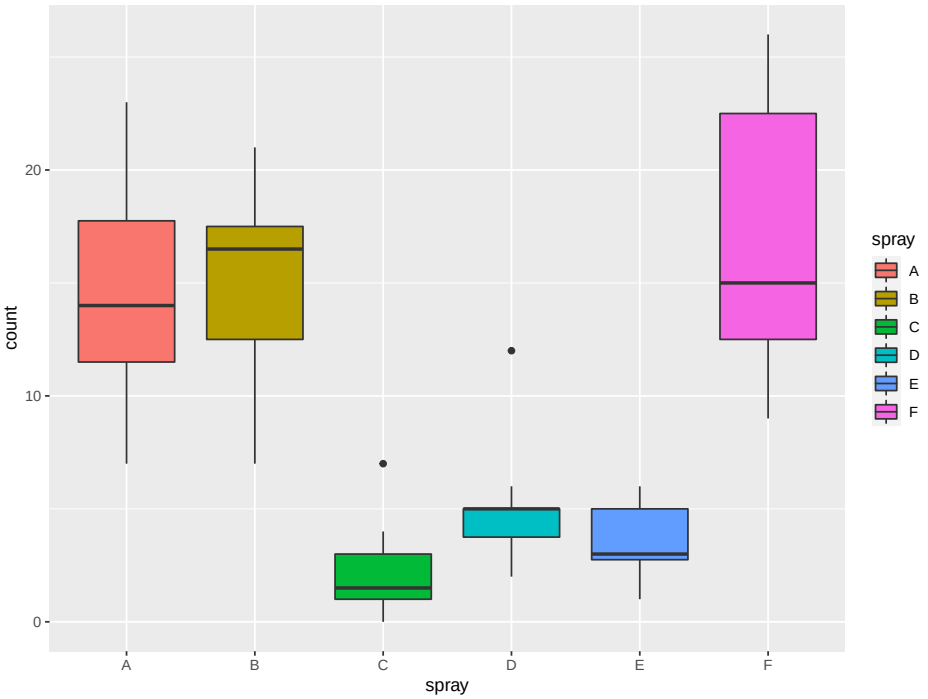
```
## [1] 0.08454034
```

```r
quantile(medians, c(.025, .975))
```

```
##     2.5%    97.5%
## 68.42972 68.81461
```

Permutation Tests

Observe the following dataset: each spray is a different pesticide, and *count* is the number of killed insects:



> The difference between B and C can be clearly seen, but a **permutation test** gives us a way to quantify it via p-values. That way we can perform a regular hypothesis test, like H_0: all sprays follow the same distribution. Permutation tests allow us to do formal inference based on an idea of exchangeability of group labels.

Notes:

- The permutation test for binary data is **Fisher's exact test**. The test for "rank sum" is the **Rank sum test**.
- Permutation strategies work for regression as well (e.g. permuting a regressor of interest)
- Permutation tests work well in multivariate settings

Procedure:

1. For 2 groups and N groups per sample, gather data in a matrix in $\mathbb{R}^{N\times 2}$.
2. Compute your desired statistic, e.g. mean difference, EV, T-statistic...
3. Repeat many times: permute the 2 elements in each row, and compute the statistic between the 2 permuted groups
4. The results of step 3 will yield a simulated distribution that assumes no difference between the 2 groups, i.e. it is our H_0 distribution.
5. We can compute the p-value of our statistic from step 2 by checking its quantile on the distribution from step 4.

Example: Going back to the spray data, we will compute the difference in means between the 2 groups:

```
subdata <- InsectSprays[InsectSprays$spray %in% c("B", "C"),]
y <- subdata$count
group <- as.character(subdata$spray)
testStat <- function(w, g) mean(w[g == "B"]) - mean(w[g == "C"])
observedStat <- testStat(y, group)
permutations <- sapply(1 : 10000, function(i) testStat(y, sample(group)))
observedStat
```

```
## [1] 13.25
```

```
mean(permutations > observedStat)
```

```
## [1] 0
```

Note the 2 output numbers: the first one is the observed mean difference of 13.25, and the second one is the number of permutations that achieved a great difference.

This means that the p-value for H_0 will be very close to zero, i.e. we can safely reject that both sprays come from the same distribution. Note that the p-value is greater than zero because there is at least one permutation that causes the difference of 13.25, that is the permutation originally seen in the dataset.

The following plot depicts the H_0 distribution and the observed statistic as a vertical line:

Histogram of permutations B v C