In this post, we seek to develop an intuitive sense of what type I (false-positive) and type II (false-negative) errors represent when comparing metrics in A/B tests, in order to gain an appreciation for “peeking”, one of the major problems plaguing the analysis of A/B test today.

To better understand what “peeking” is, it helps to first understand how to properly run a test. We will focus on the case of testing whether there is a difference between the conversion rates cr\_a and cr\_b for groups A and B. We define conversion rate as the total number of conversions in a group divided by the total number of subjects. The basic idea is that we create two experiences, A and B, and give half of the randomly-selected subjects experience A and half B. Then, after some number of users have gone through our test, we measure how many conversions happened in each group. The important question is: how many users do we need to have in groups A and B in order to measure a difference in conversion rates of a particular size?

To correctly run a test, one should first calculate the required sample size by doing a power calculation. This is easily done in R using the pwr library, which requires a few parameters: the desired significance level (the false positive rate), the desired statistical power (1-false negative rate), the minimum detectable effect, and the baseline conversion rate cr\_a.

library(pwr)

mde <- 0.1 # minimum detectable effect

cr\_a <- 0.25 # the expected conversion rate for group A

alpha <- 0.05 # the false positive rate

power <- 0.80 # 1-false negative rate

ptpt <- pwr.2p.test(h = ES.h(p1 = cr\_a, p2 = (1+mde)\*cr\_a),

sig.level = alpha,

power = power

)

n\_obs <- ceiling(ptpt$n)

This result tells us that we need to observe 4860 subjects in each of the A and B test groups if we want to detect a difference of 10% in their conversion rates. Once we have observed that quantity, we can calculate whether there is a statistically significant difference between the two sets of observations via a t-test.

Given the parameters we included in our power calculation, there are two things to be aware of:

* there is a 5% chance that our t-test will predict that there is a statistically significant difference when, in fact, there isn’t (a false positive). That is a result of our alpha parameter, which sets a false-positive rate of 5%.
* there is a 20% chance that the t-test will predict no difference when there actually was a difference (a false negative). This is the false-negative rate (or 1-power), and is commonly referred to as beta.

**Illustrating alpha and beta parameters**

Let’s try to illustrate this through a simulation. We’ll simulate two sequences of conversions with conversion rates such that cr\_b = (1+mde)\*cr\_a. We’ll then run a t-test comparing all of the available observations of the two groups **each time we have a new pair of observations**. If the p-value of the t-test is below 5%, we will reject the null-hypothesis that there is no difference between the distributions; hence, p.value < 0.05 implies there is a statistically significant difference between the conversion rates. Finally we’ll plot 1-p.value to represent “confidence” relative to where we are in the sequence. This essentially simulates what we would see **if we were to continually monitor p-values** as subjects convert or abandon. We add the 95% confidence line horizontally, as well as a vertical line at n\_obs, the number of observations our power calculation says to use to conduct a t-test.

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.5.2

set.seed(2)

# make our "true" effect larger than the mde

effect <- mde

cr\_b <- (1+effect)\*cr\_a

observations <- 2\*n\_obs

# a sequence of {0,1} conversions

conversions\_a <- rbinom(observations, 1, cr\_a)

conversions\_b <- rbinom(observations, 1, cr\_b)

# Calculate p-values at each simultaneous observation of the a and b groups

tt <- sapply(10:observations, function(x){

t.test(conversions\_a[1:x],conversions\_b[1:x])$p.value

})

tt <- data.frame(p.value = unlist(tt))

# for plots

conf\_95 <- data.frame( x = c(-Inf, Inf), y = 0.95 )

obs\_limit\_line <- data.frame( x = n\_obs, y = c(-Inf, Inf) )

# plot the evolution of p-value over time, if "peeking"

ggplot(tt, aes(x=seq\_along(p.value), y=1-p.value)) +

geom\_line() +

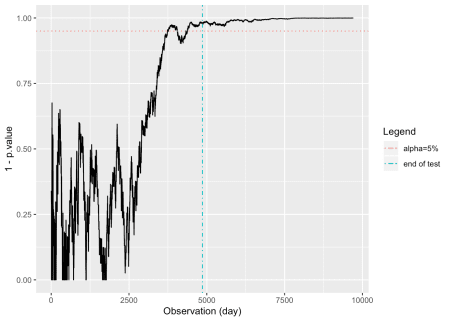
geom\_line(aes(x, y, color="alpha=5%"), linetype=3, conf\_95) +

geom\_line(aes(x, y, color="end of test"), linetype=4, obs\_limit\_line) +

xlab("Observation (day)") +

scale\_color\_discrete(name = "Legend") +

ylim(c(0,1))



We observe that in the above example, we would correctly have measured a difference in conversion rates *for this particular simulation*. However, we should expect that if we run this experiment 100 times, about 20 of those times will result in an incorrect negative prediction due to our false negative rate being 20% (power=80).

To test this, and other concepts in this article, we are going to create a utility function that will run a simulation repeatedly:

#

# monte carlo runs n\_simulations and calls the callback function each time with the ... optional args

#

monte\_carlo <- function(n\_simulations, callback, ...){

simulations <- 1:n\_simulations

sapply(1:n\_simulations, function(x){

callback(...)

})

}

Now, we’ll use the monte\_carlo utility function to run 1000 experiments, measuring whether the p.value is less than alpha **after n\_obs observations**. If it is, we reject the null hypothesis. We expect about 800 rejections and about 200 non-rejections, since 200/1000 would represent our expected 20% false negative rate.

reject\_at\_i <- function(observations, i){

conversions\_a <- rbinom(observations, 1, cr\_a)

conversions\_b <- rbinom(observations, 1, cr\_b)

( t.test(conversions\_a[1:i],

conversions\_b[1:i])$p.value ) < alpha

}

# run the sim

rejected.H0 <- monte\_carlo(1000,

callback=reject\_at\_i,

observations=n\_obs,

i=n\_obs

)

# output the rejection table

table(rejected.H0)

## rejected.H0

## FALSE TRUE

## 190 810

We will use the same functions to test the false positive rate. In this case, we want to set the two conversion rates to the same value, cr\_a, and confirm that out of 1000 experiments, about 50 show up as having rejected the null hypothesis (and predict a difference in the conversion rates).

reject\_at\_i <- function(observations, i){

conversions\_a <- rbinom(observations, 1, cr\_a)

conversions\_b <- rbinom(observations, 1, cr\_a) # this is now the same conversion rate

( t.test(conversions\_a[1:i],

conversions\_b[1:i])$p.value ) < alpha

}

# run the sim

rejected.H0 <- monte\_carlo(1000,

callback=reject\_at\_i,

observations=n\_obs,

i=n\_obs

)

# output the rejection table

table(rejected.H0)

## rejected.H0

## FALSE TRUE

## 962 38

Indeed, the results are as expected. We have shown that if we measure the results of our experiment when our power calculation tells us to, we can expect the false positive and false negative rates to reflect the values we have set in alpha and power.

**Peeking**

What happens if we don’t do a power calculation and instead monitor the conversions as they come in? This is what is termed “peeking”. In other words, to run a test correctly, you should only observe the results at one moment: when the power calculation has told you the test is complete. If you look at any moment prior to this, you are peeking at the results.

Peeking is widespread when it comes to analyzing experiments. In fact, some popular testing services peek continuously and automatically notify as soon as a p-value is below alpha. What you may notice, however, is that p-values can fluctuate around alpha multiple times before “choosing” a side, particularly where the size of an effect is small. Continuously monitoring p-values will inflate false positive rates. Let’s see what effect continuous peeking has on our false positive rates:

peeking\_method <- function (observations, by=1){

# create the conversions

conversions\_a <- rbinom(observations, 1, cr\_a)

conversions\_b <- rbinom(observations, 1, cr\_a) # no effect

reject <- FALSE;

# for each simulation, calculate the running conversion rates and days to complete.

# Break the first time we have run more than the required days to complete

for (i in seq(from=by,to=observations,by=by)) {

tryCatch(

{

reject <- ( t.test(conversions\_a[1:i],conversions\_b[1:i])$p.value ) < alpha

if(reject){

break;

}

}, error=function(e){

print(e)

}

)

}

reject

}

# run the sim

rejected.H0 <- monte\_carlo(1000,

callback=peeking\_method,

observations=n\_obs,

by=100

)

# output the rejection table

table(rejected.H0)

## rejected.H0

## FALSE TRUE

## 653 347

Indeed, even peeking every 100 sets of observations leads to an inflated false positive rate of about 30%! Peeking at test results to make decisions, and especially automating peeking to make decisions, is a big no-no.

In a subsequent article, we’ll explore an alternative way of measuring p-values that is not affected by peeking, based on sequential probability ratio tests.