This post builds upon:

Comparing Frequentist, Bayesian and Simulation methods and conclusions

I’m not here to belabor the nuances but frequentist methods are among the oldest and arguably the most prevalent method in many fields. They are often the first method people learned in college and sometimes the only method. They still drive most of the published research in many fields although other methods are taking root.

Before the frequentist can tell their tale though let’s make sure they have the same data as in the earlier post. Let’s load all the libraries we’re going to use and very quickly reproduce the process Anindya Mozumdar went through to scrape and load the data. We’ll have a tibble named male\_100 that contains the requisite data and we’ll confirm that the summary for the top 6 runners mean and median are identical. Note that I am suppressing messages as the libraries load since R 3.6.0 has gotten quite chatty in this regard.

library(rvest)

library(readr)

library(dplyr)

library(ggplot2)

library(ggstatsplot)

library(BayesFactor)

male\_100\_html <- read\_html("http://www.alltime-athletics.com/m\_100ok.htm")

male\_100\_pres <- male\_100\_html %>%

html\_nodes(xpath = "//pre")

male\_100\_htext <- male\_100\_pres %>%

html\_text()

male\_100\_htext <- male\_100\_htext[[1]]

male\_100 <- readr::read\_fwf(male\_100\_htext, skip = 1, n\_max = 3178,

col\_types = cols(.default = col\_character()),

col\_positions = fwf\_positions(

c(1, 16, 27, 35, 66, 74, 86, 93, 123),

c(15, 26, 34, 65, 73, 85, 92, 122, 132)

))

male\_100 <- male\_100 %>%

select(X2, X4) %>%

transmute(timing = X2, runner = X4) %>%

mutate(timing = gsub("A", "", timing),

timing = as.numeric(timing)) %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell", "Yohan Blake",

"Justin Gatlin", "Maurice Greene", "Tyson Gay")) %>%

mutate\_if(is.character, as.factor) %>%

droplevels

male\_100

## # A tibble: 535 x 2

## timing runner

## <dbl> <fct>

## 1 9.58 Usain Bolt

## 2 9.63 Usain Bolt

## 3 9.69 Usain Bolt

## 4 9.69 Tyson Gay

## 5 9.69 Yohan Blake

## 6 9.71 Tyson Gay

## 7 9.72 Usain Bolt

## 8 9.72 Asafa Powell

## 9 9.74 Asafa Powell

## 10 9.74 Justin Gatlin

## # … with 525 more rows

male\_100$runner <- forcats::fct\_reorder(male\_100$runner, male\_100$timing)

male\_100 %>%

group\_by(runner) %>%

summarise(mean\_timing = mean(timing)) %>%

arrange(mean\_timing)

## # A tibble: 6 x 2

## runner mean\_timing

## <fct> <dbl>

## 1 Usain Bolt 9.90

## 2 Asafa Powell 9.94

## 3 Tyson Gay 9.95

## 4 Justin Gatlin 9.96

## 5 Yohan Blake 9.97

## 6 Maurice Greene 9.97

male\_100 %>%

group\_by(runner) %>%

summarise(median\_timing = median(timing)) %>%

arrange(median\_timing)

## # A tibble: 6 x 2

## runner median\_timing

## <fct> <dbl>

## 1 Usain Bolt 9.89

## 2 Asafa Powell 9.95

## 3 Justin Gatlin 9.97

## 4 Maurice Greene 9.97

## 5 Tyson Gay 9.97

## 6 Yohan Blake 9.98

Most of the code above is simply shortened from the original post. The only thing that is completely new is forcats::fct\_reorder(male\_100$runner, male\_100$timing) which takes the runner factor and reorders it according to the median by runner. This doesn’t matter for the calculations we’ll do but it will make the plots look nicer.

**Testing, testing!**

One of the issues with a frequentist approach compared to a programmers approach is that there are a lot of different tests you could choose. But in this case wearing my frequentist hat there really are only two choices. A Oneway ANOVA or the Kruskall Wallis which uses ranks and eliminates some assumptions.

This also gives me a chance to talk about a great package that supports both frequentists and bayesian methods and completely integrates visualizing your data with analyzing your data, which IMHO is the only way to go. The package is ggstatsplot. Full disclosure I’m a minor contributor to the package but please know that the true hero of the package is Indrajeet Patil. It’s stable, mature, well tested and well maintained – try it out.

So let’s assume we want to run a classic Oneway ANOVA first (Welch’s method so we don’t have to assume equal variances across groups). Assuming that the omnibuds F test is significant lets say we’d like to look at the pairwise comparisons and adjust the p values for multiple comparison using Holm. We’re a big fan of visualizing the data by runner and of course we’d like to plot things like the mean and median and the number of races per runner. We’d of course like to know our effect size we’ll stick with eta squared we’d like it as elegant as possible.

Doing this analysis using frequentist methods in R is not difficult. Heck I’ve even blogged about it myself it’s so “easy”. The benefit of ggbetweenstats from ggstatsplot is that it pretty much allows you to do just about everything in one command. Seamlessly mixing the plot and the results into one output. We’re only going to scratch the surface of all the customization possibilities.

ggbetweenstats(data = male\_100,

x = runner,

y = timing,

type = "p",

var.equal = FALSE,

pairwise.comparisons = TRUE,

partial = FALSE,

effsize.type = "biased",

point.jitter.height = 0,

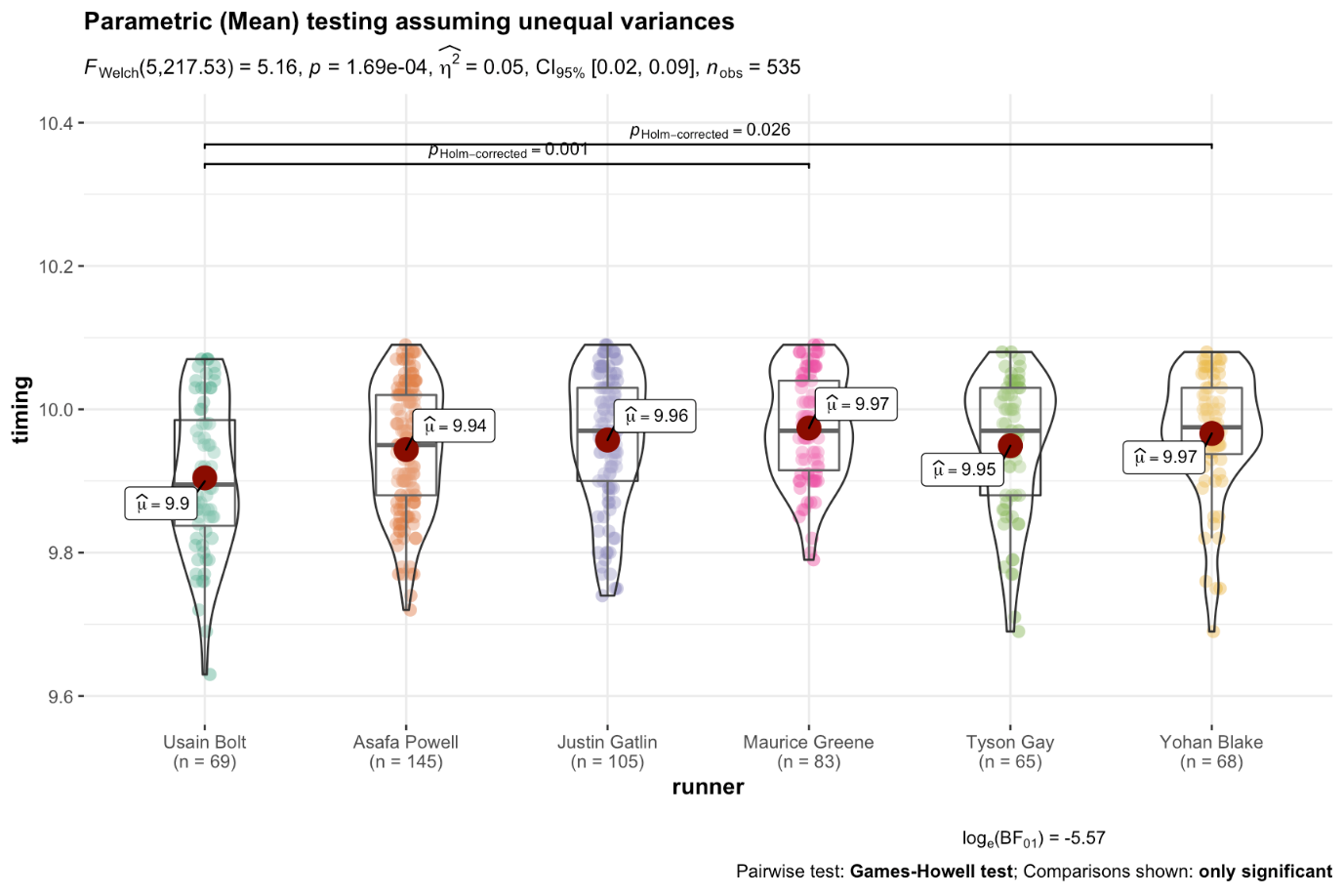
title = "Parametric (Mean) testing assuming unequal variances",

ggplot.component = ggplot2::scale\_y\_continuous(breaks = seq(9.6, 10.4, .2),

limits = (c(9.6,10.4))),

messages = FALSE

)



Our conclusion is similar to that drawn by simulation. We can clearly reject the null that all these runners have the same mean time. Using Games-Howell and controlling for multiple comparisons with Holm, however, we can only show support for the difference between Usain Bolt and Maurice Green. There is insufficient evidence to reject the null for all the other possible pairings. (You can actually tell ggbetweenstats to show the p value for all the pairings but that gets cluttered quickly).

From a frequentist perspective there are also a whole set of non-parametric tests that are available for use. They typically make fewer assumptions about the data we have and often operate by exchanging the ranks of the outcome variable (timing) rather than using the number.

The only thing we need to change in our input to the function is type = "np" and the title.

ggbetweenstats(data = male\_100,

x = runner,

y = timing,

type = "np",

var.equal = FALSE,

pairwise.comparisons = TRUE,

partial = FALSE,

effsize.type = "biased",

point.jitter.height = 0,

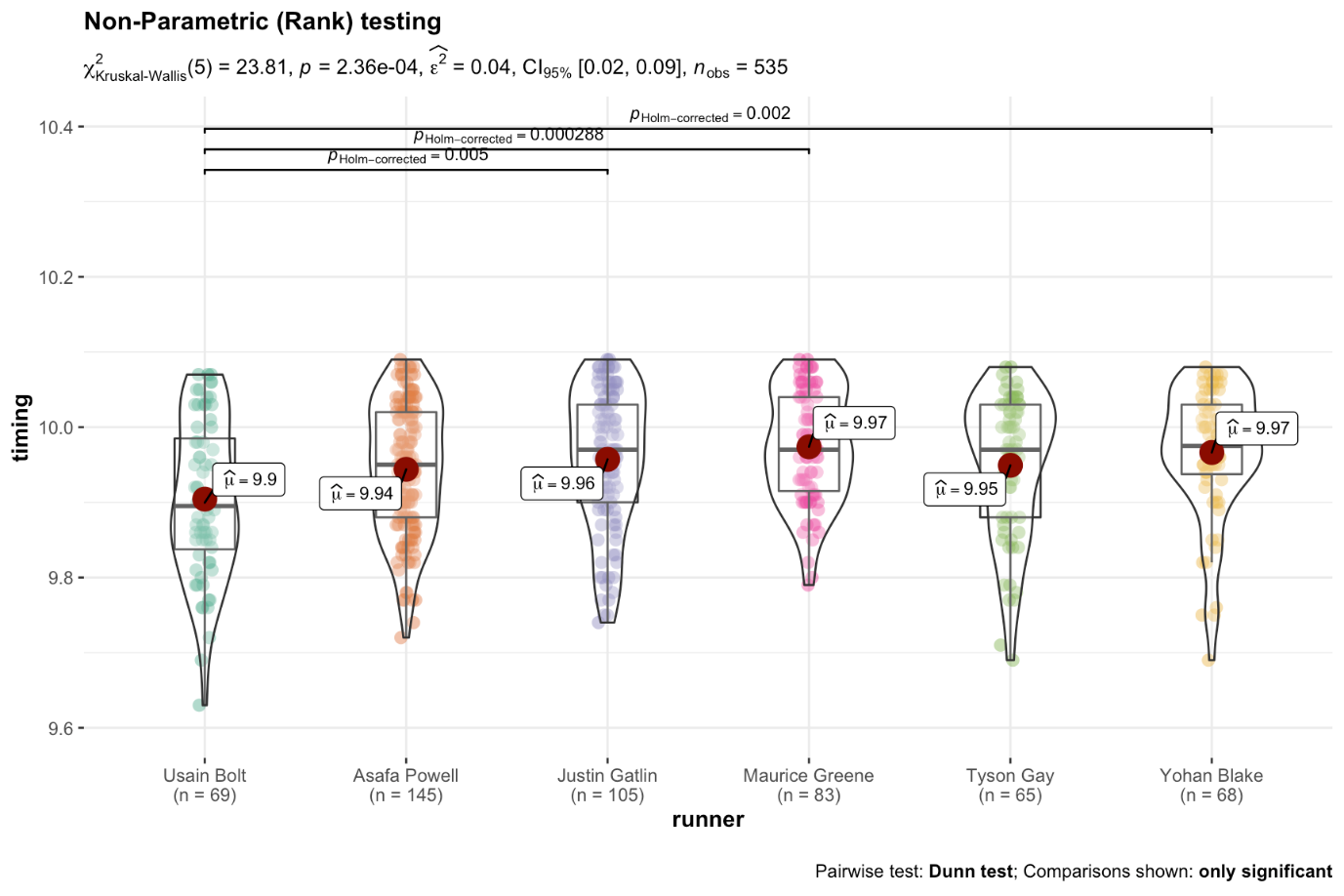
title = "Non-Parametric (Rank) testing",

ggplot.component = ggplot2::scale\_y\_continuous(breaks = seq(9.6, 10.4, .2),

limits = (c(9.6,10.4))),

messages = FALSE

)



Without getting overly consumed by the exact numbers note the very similar results for the overall test, but that we now also are more confident about whether the difference between Usain Bolt and Justin Gaitlin. I highlight that because there is a common misconception that non-parametric tests are always less powerful (sensitive) than their parametric cousins.

**Asking the question differently (see Learning Statistics with R )**

*Much of the credit for this section goes to Danielle Navarro (bookdown translation: Emily Kothe) in Learning Statistics with R*

It usually takes several lessons or even an entire semester to teach the frequentist method, because null hypothesis testing is a very elaborate contraption that people (well in my experience very smart undergraduate students) find very hard to master. In contrast, the Bayesian approach to hypothesis testing *“feels”* far more intuitive. Let’s apply it to our current scenario.

We’re at the bar the three of us wondering whether Usain Bolt is really the fastest or whether all these individual data points really are just a random mosaic of data noise. Both the programmer and the frequentist set the testing up conceptually the same way. Can we use the data to reject the null that all the runners are the same. Convinced they’re not all the same they applied the same general procedure to reject (or not) the hypothesis that any pair was the same for example Bolt versus Powell (for the record I’m not related to either). They differ in computational methods and assumptions but not in overarching method.

At the end of their machinations they have no ability to talk about how likely (probable) it is that runner 1 will beat runner 2. Often times that’s exactly what you really want to know. There are two hypotheses that we want to compare, a null hypothesis h0 that all the runners run equally fast and an alternative hypothesis h1 that they don’t. Prior to looking at the data while we’re sitting at the bar we have no real strong belief about which hypothesis is true (odds are 1:1 in our naive state). We have our data and we want it to inform our thinking. Unlike frequentist statistics, Bayesian statistics allow us to talk about the probability that the null hypothesis is true (which is a complete **no no** in a frequentist context). Better yet, it allows us to calculate the posterior probability of the null hypothesis, using Bayes’ rule and our data.

In practice, most Bayesian data analysts tend not to talk in terms of the raw posterior probabilities. Instead, we/they tend to talk in terms of the posterior odds ratio. Think of it like betting. Suppose, for instance, the posterior probability of the null hypothesis is 25%, and the posterior probability of the alternative is 75%. The alternative hypothesis **h1** is three times as probable as the null **h0**, so we say that the odds are 3:1 in favor of the alternative.

At the end of the Bayesian’s efforts they can make what feel like very natural statements of interest, for example, "The evidence provided by our data corresponds to odds of 42:1 that these runners are not all equally fast.

Let’s try it using ggbetweenstats…

ggbetweenstats(data = male\_100,

x = runner,

y = timing,

type = "bf",

var.equal = FALSE,

pairwise.comparisons = TRUE,

partial = FALSE,

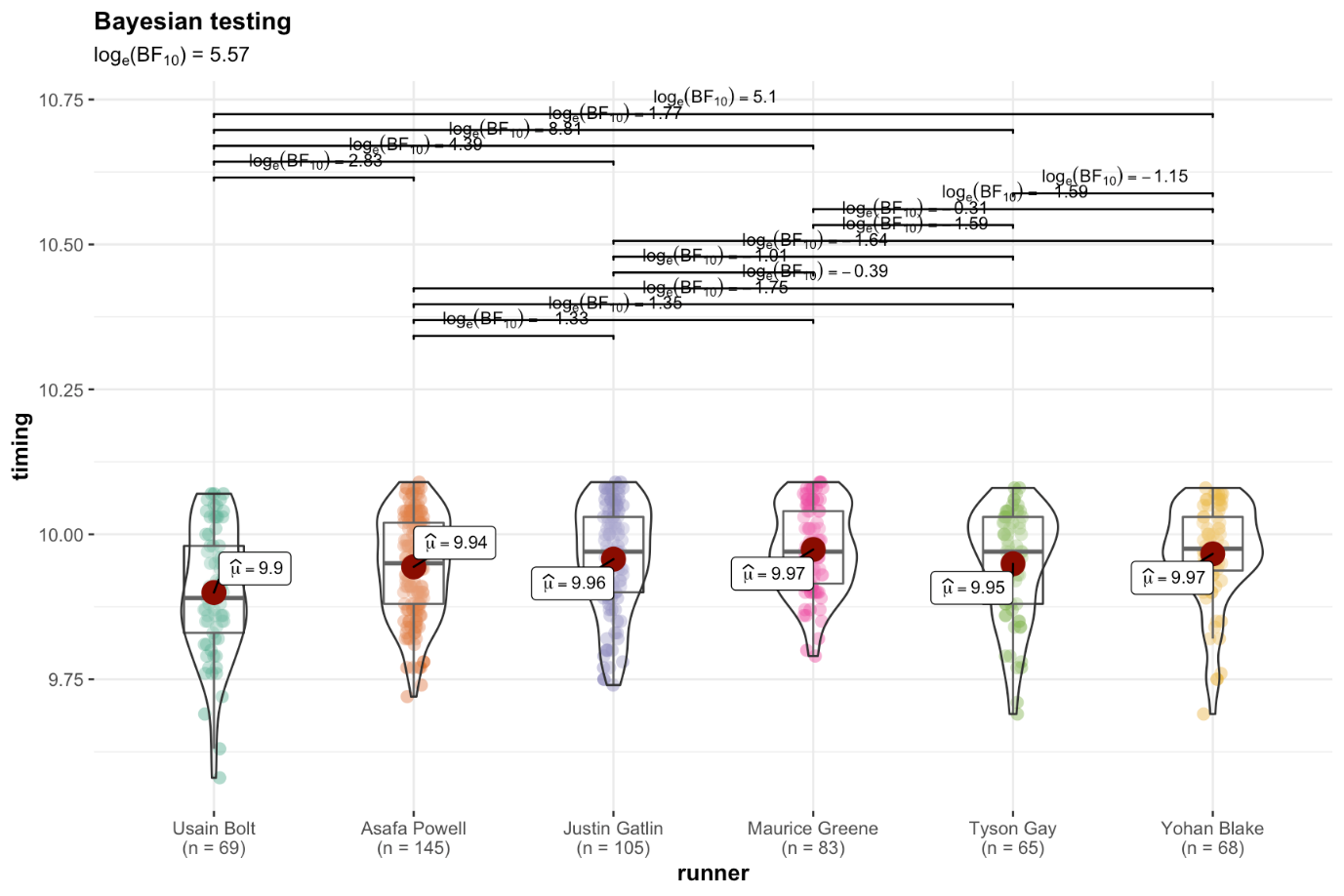
effsize.type = "biased",

point.jitter.height = 0,

title = "Bayesian testing",

messages = FALSE

)



Yikes! Not what I wanted to see in the bar. The pairwise comparisons have gone away (we’ll get them back) and worse yet what the heck does loge(BF10) = 2.9 mean? I hate log conversions I was promised a real number like 42:1! Who’s Cauchy why is he there at .0.707?

Let’s break this down. loge(BF10) = 2.9 is also exp(2.9) or about 18 so the good news is the odds are better than 18:1 that the runners are not equally fast. Since rounding no doubt loses some accuracy lets use the BayesFactor package directly and get a more accurate answer before we round anovaBF(timing ~ runner, data = as.data.frame(male\_100), rscaleFixed = .707) is what we want where rscaleFixed = .707 ensures we have the right Cauchy value.

anovaBF(timing ~ runner, data = male\_100, rscaleFixed = .707)

## Bayes factor analysis

## --------------

## [1] runner : 263.6038 ±0%

##

## Against denominator:

## Intercept only

## ---

## Bayes factor type: BFlinearModel, JZS

Okay that’s better so to Bayesian thinking the odds are 19:1 against the fact that they all run about the same speed, or 19:1 they run at different speeds.

Hmmm. One of the strengths/weaknesses of the Bayesian approach is that people can have their own sense of how strong 19:1 is. I like those odds. One of the really nice things about the Bayes factor is the numbers are inherently meaningful. If you run the data and you compute a Bayes factor of 4, it means that the evidence provided by your data corresponds to betting odds of 4:1 in favor of the alternative. However, there have been some attempts to quantify the standards of evidence that would be considered meaningful in a scientific context. One that is widely used is from Kass and Raftery (1995). (**N.B. there are others and I have deliberately selected one of the most conservative standards. See, for example, this very nice graphic with the evidentiary standard attributed to Wagenmakers, Wetzels, Borsboom, & Van Der Maas, 2011 with graphic from Springer Nature.**)

| **Bayes factor value** | **Interpretation** |
| --- | --- |
| 1 - 3 | Negligible evidence |
| 3 - 20 | Positive evidence |
| 20 -150 | Strong evidence |
| >150 | Very strong evidence |

Okay we have “positive evidence” and we can quantify it, that’s good. But what about all the pairwise comparisons? Can we take this down to all the individual pairings? I’m on the edge of my bar stool here. What are the odds Bolt really is faster than Powell? Can we quantify that without somehow breaking the multiple comparisons rule?

The short answer is yes we can safely extend this methodology to incorporate pairwise comparisons. We shouldn’t abuse the method and we should fit our model with the best possible prior information but in general, as simulated here,

*With Bayesian inference (and the correct prior), though, this problem disappears. Amazingly enough, you don’t have to correct Bayesian inferences for multiple comparisons.*

With that in mind let’s build a quick little function that will allow us to pass a data source and two names and run a Bayesian t-test via BayesFactor::ttestBF to compare two runners. ttestBF returns a lot of info in a custom object so we’ll use the extractBF function to grab it in a format where we can pluck out the actual BF10

compare\_runners\_bf <- function(df, runner1, runner2) {

ds <- df %>%

filter(runner %in% c(runner1, runner2)) %>%

droplevels %>%

as.data.frame

zzz <- ttestBF(formula = timing ~ runner, data = ds)

yyy <- extractBF(zzz)

xxx <- paste0("The evidence provided by the data corresponds to odds of ",

round(yyy$bf,0),

":1 that ",

runner1,

" is faster than ",

runner2 )

return(xxx)

}

Now that we have a function we can see what the odds are that Bolt is faster than the other 5 and print them one by one

compare\_runners\_bf(male\_100, "Usain Bolt", "Asafa Powell")

## [1] "The evidence provided by the data corresponds to odds of 17:1 that Usain Bolt is faster than Asafa Powell"

compare\_runners\_bf(male\_100, "Usain Bolt", "Tyson Gay")

## [1] "The evidence provided by the data corresponds to odds of 6:1 that Usain Bolt is faster than Tyson Gay"

compare\_runners\_bf(male\_100, "Usain Bolt", "Justin Gatlin")

## [1] "The evidence provided by the data corresponds to odds of 80:1 that Usain Bolt is faster than Justin Gatlin"

compare\_runners\_bf(male\_100, "Usain Bolt", "Yohan Blake")

## [1] "The evidence provided by the data corresponds to odds of 163:1 that Usain Bolt is faster than Yohan Blake"

compare\_runners\_bf(male\_100, "Usain Bolt", "Maurice Greene")

## [1] "The evidence provided by the data corresponds to odds of 6720:1 that Usain Bolt is faster than Maurice Greene"

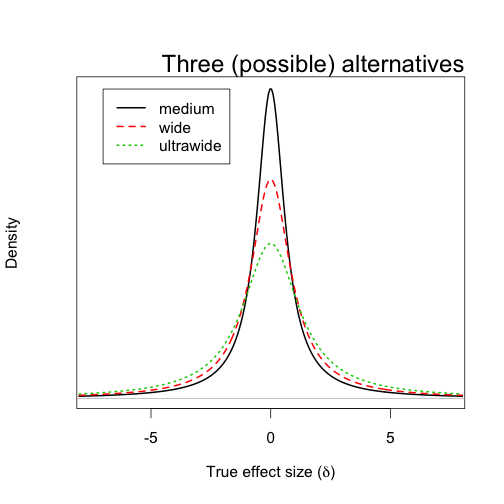
Okay now I feel like we’re getting somewhere with our bar discussions. Should I feel inclined to make a little wager on say who buys the next round of drinks as a Bayesian I have some nice useful information. I’m not rejecting a null hypothesis I’m casting the information I have as a statement of the odds I think I have of “winning”.

But of course this isn’t the whole story so please read on…

**Who’s Cauchy and why does he matter?**

Earlier I made light of the fact that the output from ggbetweenstats had rCauchy = 0.707 and anovaBF uses rscaleFixed = .707. Now we need to spend a little time actually understanding what that’s all about. Cauchy is Augustin-Louis Cauchy and the reason that’s relevant is that BayesFactor makes use of his distribution as a default. I’m not even going to try and take you into the details of the math but it is important we have a decent understanding of what we’re doing to our data.

The BayesFactor package has a few built-in “default” named settings. They all have the same shape; the only differ by their scale, denoted by r. The three named defaults are medium = 0.707, wide = 1, and ultrawide = 1.414. “Medium”, is the default. The scale controls how large, on average, the expected true effect sizes are. For a particular scale 50% of the true effect sizes are within the interval (−r, r). For the default scale of “medium”, 50% of the prior effect sizes are within the range (−0.7071, 0.7071). Increasing r increases the sizes of expected effects; decreasing r decreases the size of the expected effects.



BayesFactor blog site – February 23, 2014

Let’s compare it to a frequentist test we’re all likely to know, the t-test, (we’ll use the Welch variant). Our initial hypothesis is that Bolt’s mean times are different than Powell’s mean times (two-sided) and then test the one-sided that Bolt is faster. Then let’s go a little crazy and run it one sided but specify the mean difference 0.038403 of a second faster that we “see” in our data mu = -0.038403.

justtwo <- male\_100 %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell")) %>%

droplevels %>%

as.data.frame

t.test(formula = timing ~ runner, data = justtwo)

##

## Welch Two Sample t-test

##

## data: timing by runner

## t = -2.9288, df = 109.17, p-value = 0.004143

## alternative hypothesis: true difference in means is not equal to 0

## 95 percent confidence interval:

## -0.0742729 -0.0143208

## sample estimates:

## mean in group Usain Bolt mean in group Asafa Powell

## 9.899565 9.943862

t.test(formula = timing ~ runner, data = justtwo, alternative = "less")

##

## Welch Two Sample t-test

##

## data: timing by runner

## t = -2.9288, df = 109.17, p-value = 0.002072

## alternative hypothesis: true difference in means is less than 0

## 95 percent confidence interval:

## -Inf -0.01920609

## sample estimates:

## mean in group Usain Bolt mean in group Asafa Powell

## 9.899565 9.943862

t.test(formula = timing ~ runner, data = justtwo, alternative = "less", mu = -0.038403)

##

## Welch Two Sample t-test

##

## data: timing by runner

## t = -0.38968, df = 109.17, p-value = 0.3488

## alternative hypothesis: true difference in means is less than -0.038403

## 95 percent confidence interval:

## -Inf -0.01920609

## sample estimates:

## mean in group Usain Bolt mean in group Asafa Powell

## 9.899565 9.943862

Hopefully that last one didn’t trip you up and you recognized by definition if the mean difference in our sample data is -0.038403 then the p value should reflect 50/50 p value?

Let’s first just try different rCauchy values with ttestBF.

justtwo <- male\_100 %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell")) %>%

droplevels %>%

as.data.frame

ttestBF(formula = timing ~ runner, data = justtwo, rscale = "medium")

## Bayes factor analysis

## --------------

## [1] Alt., r=0.707 : 16.99936 ±0%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

ttestBF(formula = timing ~ runner, data = justtwo, rscale = "wide")

## Bayes factor analysis

## --------------

## [1] Alt., r=1 : 14.0425 ±0%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

ttestBF(formula = timing ~ runner, data = justtwo, rscale = .2)

## Bayes factor analysis

## --------------

## [1] Alt., r=0.2 : 16.55976 ±0%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

Okay the default medium returns just what we reported earlier 5:1 odds. Going wider gets us 4:1 and going narrower (believing the difference is smaller) takes us to 6:1. Not huge differences but noticeable and driven by our data.

Let’s investigate directional hypotheses with ttestBF. First let’s ask what’s the evidence that Bolt is faster than Powell **NB the order is driven by factor level in the dataframe not the order in the filter command below. Also note that faster is a lower number**

justtwo <- male\_100 %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell")) %>%

droplevels %>%

as.data.frame

# notice these two just return the same answer in a different order

ttestBF(formula = timing ~ runner, data = justtwo, nullInterval = c(0, Inf))

## Bayes factor analysis

## --------------

## [1] Alt., r=0.707 0<d<Inf : 0.03861953 ±0.01%

## [2] Alt., r=0.707 !(0<d<Inf) : 33.9601 ±0%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

ttestBF(formula = timing ~ runner, data = justtwo, nullInterval = c(-Inf, 0))

## Bayes factor analysis

## --------------

## [1] Alt., r=0.707 -Inf<d<0 : 33.9601 ±0%

## [2] Alt., r=0.707 !(-Inf<d<0) : 0.03861953 ±0.01%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

So the odds that Bolt has a bigger number i.e. is slower than Powell is 0.04:1 and the converse is the odds that Bolt has a smaller timing (is faster) is 10:1. You can feel free to put these in the order that makes the most sense to your workflow. They’re always going to be mirror images.

And yes in some circumstances it is perfectly rational to combine the information by dividing those odds. See this blog post for a research scenario but accomplishing it is trivial. Running this code snippet essentially combines what we know in both directions of the hypothesis.

justtwo <- male\_100 %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell")) %>%

droplevels %>%

as.data.frame

powellvbolt <- ttestBF(formula = timing ~ runner, data = justtwo, nullInterval = c(-Inf, 0))

powellvbolt[1]/powellvbolt[2]

## Bayes factor analysis

## --------------

## [1] Alt., r=0.707 -Inf<d<0 : 879.3505 ±0.01%

##

## Against denominator:

## Alternative, r = 0.707106781186548, mu =/= 0 !(-Inf<d<0)

## ---

## Bayes factor type: BFindepSample, JZS

More Bayes and multiple comparisons

Let’s quickly create our dataset following the same methodology as the last two times, with the important exception that we’re going to focus on only the 3 fastest runners. Everything I’m going to do is extensible beyond that, but for simplicity sake we’ll limit ourselves to just three runners. We’ll load the required packages (suppressing the chatty load process) and create our dataset, this time named best3 to reflect the fact we’ve reduced ourselves down to the 3 runners with the fastest mean time, Usain Bolt, Asafa Powell, and Yohan Blake. So we have a tibble with 273 rows with the runner’s name and how fast they ran.

library(rvest)

library(readr)

library(tidyverse)

library(ggstatsplot)

library(jmv)

library(BayesFactor)

library(gtools)

library(kableExtra)

male\_100\_html <- read\_html("http://www.alltime-athletics.com/m\_100ok.htm")

male\_100\_pres <- male\_100\_html %>%

html\_nodes(xpath = "//pre")

male\_100\_htext <- male\_100\_pres %>%

html\_text()

male\_100\_htext <- male\_100\_htext[[1]]

male\_100 <- read\_fwf(male\_100\_htext, skip = 1, n\_max = 3178,

col\_types = cols(.default = col\_character()),

col\_positions = fwf\_positions(

c(1, 16, 27, 35, 66, 74, 86, 93, 123),

c(15, 26, 34, 65, 73, 85, 92, 122, 132)

))

best3 <- male\_100 %>%

select(X2, X4) %>%

transmute(timing = X2, runner = X4) %>%

mutate(timing = gsub("A", "", timing),

timing = as.numeric(timing)) %>%

filter(runner %in% c("Usain Bolt", "Asafa Powell", "Yohan Blake")) %>%

mutate\_if(is.character, as.factor) %>%

droplevels

best3$runner <- fct\_reorder(best3$runner, best3$timing)

best3

## # A tibble: 282 x 2

## timing runner

## <dbl> <fct>

## 1 9.58 Usain Bolt

## 2 9.63 Usain Bolt

## 3 9.69 Usain Bolt

## 4 9.69 Yohan Blake

## 5 9.72 Usain Bolt

## 6 9.72 Asafa Powell

## 7 9.74 Asafa Powell

## 8 9.75 Yohan Blake

## 9 9.75 Yohan Blake

## 10 9.76 Usain Bolt

## # … with 272 more rows

### The frequentist and multiple comparisons

Last time I chose to look at both parametric and non parametric versions of the oneway analysis of variance. For brevity’s sake this time I am only going to run the parametric version so I can focus on Bayesian methods in a later section. Once again I’m going to use ggstatsplot::ggbetweenstats to combine a whole bunch of processes that you could also do in base R. The difference is ggbetweenstats will quickly and efficiently give us not only the tests we need but let us visualize the data at the same time. We’ll run the required omnibuds F test to determine whether we can reject the null **h0** that all the runners have the same mean time. Which is the required first step in **nhst** when you have more than two factor levels in your dependent variable. At the same time we’ll use the parameters pairwise.comparisons = TRUE and pairwise.display = "all" to display graphically our ability to reject the **h0** that all of the three possible pairings are different. This is in fact actually what we are most likely interested in for our research. Just for completeness we’ll follow-up with the base command pairwise.t.test to show the pairwise comparisons.

ggbetweenstats(data = best3,

x = runner,

y = timing,

type = "p",

var.equal = TRUE,

pairwise.comparisons = TRUE,

pairwise.display = "all",

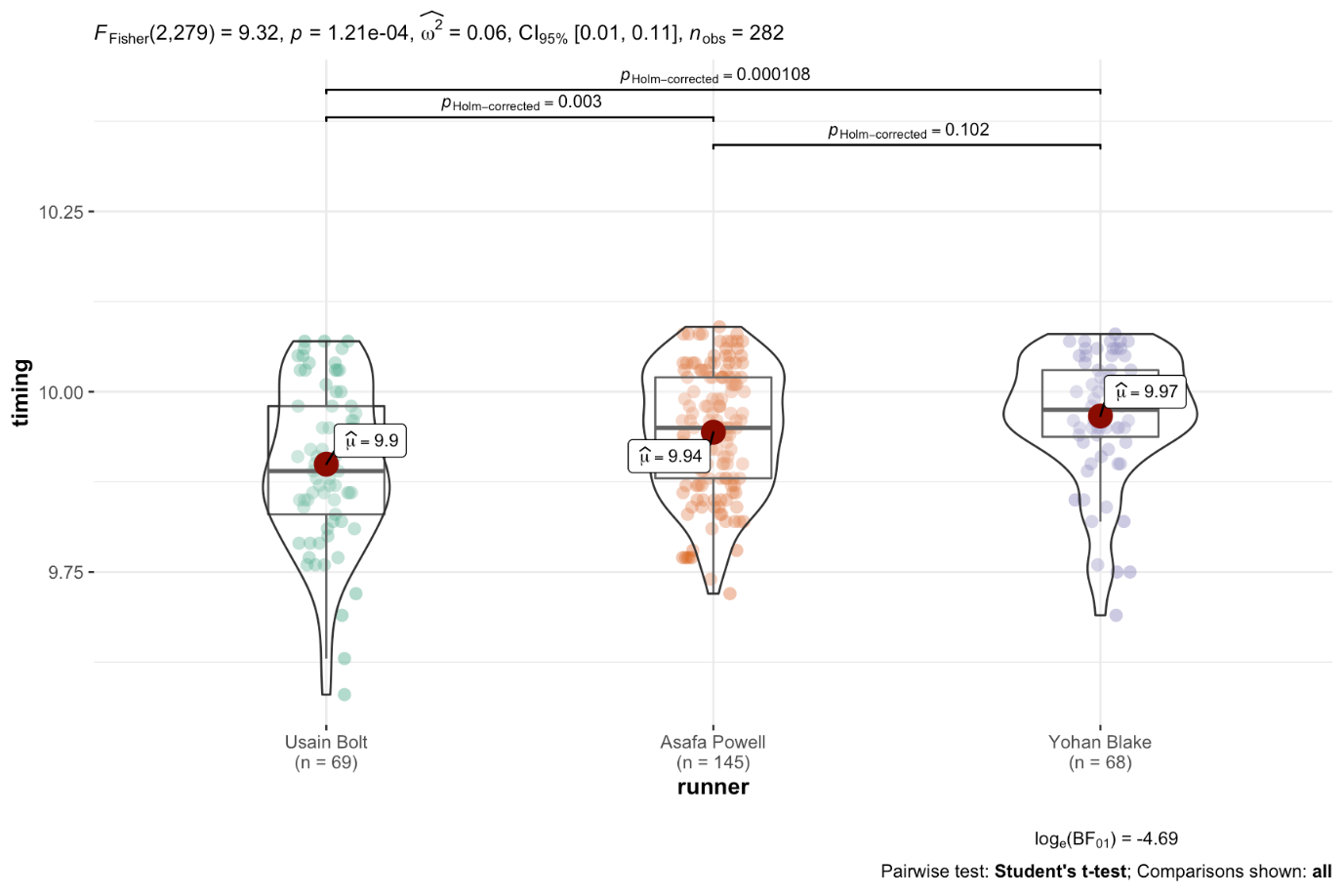
partial = FALSE,

effsize.type = "unbiased",

point.jitter.height = 0,

messages = FALSE

)



pairwise.t.test( x = best3$timing, # outcome variable

g = best3$runner, # grouping variable

p.adjust.method = "holm" # which correction to use?

)

##

## Pairwise comparisons using t tests with pooled SD

##

## data: best3$timing and best3$runner

##

## Usain Bolt Asafa Powell

## Asafa Powell 0.00255 -

## Yohan Blake 0.00011 0.10163

##

## P value adjustment method: holm

You’ve no doubt noticed that **holm** appears in both versions of the output. And that’s where we want to focus for now. The Holm method or correction is one of **many** (and by many I do mean **many**) ways of correcting for the fact that as we make more and more comparisons we increase the chances that we’ll reject the null when we shouldn’t.

#### Key take-aways

Whole books can and have been written just on the issue of multiple comparisons or simultaneous inference so I’m not going to try and summarize it all in one little blog post. I will however, for the impatient reader, summarize a few key things:

1. Planned versus unplanned comparisons matter in principle. In the literature you’ll see a clear distinction between making comparisons based upon clear hypothesis you’ve specified a prior versus things done post hoc. You are traditionally granted far more leeway if you’ve specified a small number of contrasts of interest in advance versus entered into a fishing expedition after the data are collected. Note that the math doesn’t change.
2. The quantity of comparisons matters a great deal. Every additional pairwise comparison increases your risk of making a Type I error (rejecting **h0** when you shouldn’t). From a frequentist perspective this is very bad since the entire methodology is based upon statements about controlling for type I errors.
3. Being more conservative about controlling against Type I error comes at the cost of risking more type II errors, failing to reject **h0** when you should.
4. In principle there are now two distinct approaches to multiple comparisons, the traditional Family Wise Error Rate approach which you’ll see in base R with names like "holm", "hochberg", "hommel", "bonferroni". Then there are relatively newer methods such as “BH” (Benjamini & Hochberg (1995)), and “BY” (Benjamini & Yekutieli (2001)) which control for False Discovery Rate.

Okay enough theory, let’s use our current data about the best 3 runners and see what happens across a selection of these methods.

#### What does that look like with our data

First off, we need to get the paired comparisons in some useful format like a data frame that we can work with. Enter the jmv package. It has an ANOVA (note the capitalization) function that we can make use of:

jmv::ANOVA(formula = timing ~ runner,

data=best3,

postHoc = ~runner,

postHocCorr = "none")

##

## ANOVA

##

## ANOVA - timing

## ──────────────────────────────────────────────────────────────────────────────

## Sum of Squares df Mean Square F p

## ──────────────────────────────────────────────────────────────────────────────

## runner 0.1614084 2 0.080704197 9.321641 0.0001206

## Residuals 2.4155051 279 0.008657724

## ──────────────────────────────────────────────────────────────────────────────

##

##

## POST HOC TESTS

##

## Post Hoc Comparisons - runner

## ────────────────────────────────────────────────────────────────────────────────────────────────────────────

## runner runner Mean Difference SE df t p

## ────────────────────────────────────────────────────────────────────────────────────────────────────────────

## Usain Bolt - Asafa Powell -0.04429685 0.01360819 279.0000 -3.255162 0.0012730

## - Yohan Blake -0.06675831 0.01589949 279.0000 -4.198771 0.0000361

## Asafa Powell - Yohan Blake -0.02246146 0.01367582 279.0000 -1.642422 0.1016291

## ────────────────────────────────────────────────────────────────────────────────────────────────────────────

Very nice looking, but all I really want is the post hoc section and I want it as a dataframe not in pretty printed format. Turns out anovaresults$postHoc[[1]]$asDF will give me that.

anovaresults <- ANOVA(formula = timing ~ runner,

data=best3,

postHoc = ~runner,

postHocCorr = "none")

paired\_results <- anovaresults$postHoc[[1]]$asDF

paired\_results

## runner1 sep runner2 md se df t

## 1 Usain Bolt - Asafa Powell -0.04429685 0.01360819 279 -3.255162

## 2 Usain Bolt - Yohan Blake -0.06675831 0.01589949 279 -4.198771

## 3 Asafa Powell - Yohan Blake -0.02246146 0.01367582 279 -1.642422

## pnone

## 1 1.273009e-03

## 2 3.612892e-05

## 3 1.016291e-01

Armed with that I can proceed to clean it up a bit and do some relabeling…

paired\_results <- paired\_results %>%

select(-df) %>%

mutate(Pair = str\_c(runner1,sep,runner2)) %>%

rename(meandif = md, stderr = se, tvalue = t) %>%

mutate\_at(vars(meandif, stderr, tvalue), round, 3) %>%

select(Pair, meandif, stderr, tvalue, pnone)

paired\_results

## Pair meandif stderr tvalue pnone

## 1 Usain Bolt-Asafa Powell -0.044 0.014 -3.255 1.273009e-03

## 2 Usain Bolt-Yohan Blake -0.067 0.016 -4.199 3.612892e-05

## 3 Asafa Powell-Yohan Blake -0.022 0.014 -1.642 1.016291e-01

Pair is self-explanatory and pnone contains the unadjusted probability of the t-test for that pairing. What we want to do next is to append the results for the various adjustment methods onto paired\_results. p.adjust.methods contains a vector with the names we have to choose from. Since we already have the value for **none** let’s leave that out, and since **fdr** and **BH** are identical let’s remove fdr. Armed with our list of methods we’d like to try we can use sapply to iterate through our uncorrected p values and produce a nice matrix with rounded values and store them in p.adj.

p.adjust.methods

## [1] "holm" "hochberg" "hommel" "bonferroni" "BH"

## [6] "BY" "fdr" "none"

p.adjust.M <- p.adjust.methods[p.adjust.methods %in% c("bonferroni",

"holm",

"hochberg",

"hommel",

"BH",

"BY")]

p.adj <- sapply(p.adjust.M,

function(meth) round(p.adjust(paired\_results$pnone, meth), 3))

p.adj

## holm hochberg hommel bonferroni BH BY

## [1,] 0.003 0.003 0.003 0.004 0.002 0.004

## [2,] 0.000 0.000 0.000 0.000 0.000 0.000

## [3,] 0.102 0.102 0.102 0.305 0.102 0.186

Then we can graft those values onto the paired\_results we have, do a little more cleanup and voila we have a nice table that allows us to look at all the data. I’ve taken the liberty of adding a header row to distinguish between FWER and FDR methods.

pairs\_compare <- cbind(paired\_results, as.data.frame(p.adj))

pairs\_compare <- pairs\_compare %>%

mutate(p.unadj = round(pnone,3)) %>%

select(-pnone)

# pairs\_compare # skip ordinary output in favor of nicer kable table

kable(pairs\_compare, "html") %>%

kable\_styling(bootstrap\_options = c("striped", "hover", "condensed", "responsive")) %>%

add\_header\_above(c(" " = 4, "FWER" = 4, "FDR" = 2, "None" = 1), bold = TRUE, italic = TRUE)

|  | | | | ***FWER*** | | | | ***FDR*** | | ***None*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pair** | **meandif** | **stderr** | **tvalue** | **holm** | **hochberg** | **hommel** | **bonferroni** | **BH** | **BY** | **p.unadj** |
| Usain Bolt-Asafa Powell | -0.044 | 0.014 | -3.255 | 0.003 | 0.003 | 0.003 | 0.004 | 0.002 | 0.004 | 0.001 |
| Usain Bolt-Yohan Blake | -0.067 | 0.016 | -4.199 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Asafa Powell-Yohan Blake | -0.022 | 0.014 | -1.642 | 0.102 | 0.102 | 0.102 | 0.305 | 0.102 | 0.186 | 0.102 |

Notice that for our data, the choice of method doesn’t lead to any different conclusion. We can always reject the null that Bolt vs Blake or Bolt vs Powell is equal. And no method leads us to be able to reject the possibility that Powell vs Blake is equal.

More importantly though, we’re probably not really answering the most important question(s).

### Asking different questions, differently

As I mentioned last post I’m becoming both more comfortable with using Bayesian methods and more convinced they do a better job of answering the research questions we’re really interested in. Instead of rejecting the null as a yes/no decision at some level of **\(\alpha\)** = .05, .01 or .001 the Bayesian wants to know what are the odds, given our data, that my research question **h1** is supported. So we can ask the same question of the oneway anova with…

bf1 <- anovaBF(timing ~ runner, data = best3)

bf1

## Bayes factor analysis

## --------------

## [1] runner : 144.1756 ±0.03%

##

## Against denominator:

## Intercept only

## ---

## Bayes factor type: BFlinearModel, JZS

Which allows us to say the evidence provided by the data corresponds to odds of 7:1 that the runners are not equally fast. As I mentioned last post that is actually a marked improvement in and of itself. But wait there’s more. Obviously we’re not really interested in a blanket statement about at least one being faster than the others we want to know how they compare at the paired level, analogous to paired comparisons for a frequentist.

One of the other nice features about a Bayesian approach is that we don’t have to worry nearly as much about the multiple comparisons issue Gelman, Hill, Yajima (2012). So let’s use the ttestBF function to calculate the BF for each of our 3 pairings in question and append them to our existing dataframe pairs\_compare.

**N.B. Before anyone comments yes the code below absolutely screams out asking to be turned into one of more functions. And I will, one of these days, but for now please permit me some laziness. Or if it truly offends you feel free to write the code yourself and send it to me.**

We’ll create an empty vector to hold our 3 Bayes Factors we’re going to calculate. We’ll take best3 and filter out for just the pair we want. Run ttestBF. Extract just the Bayes Factor using extractBF %>% .$bf and put it in the vector. Add a column to the dataframe rounding as we go pairs\_compare$BFfactor <- round(bfpaired,1). Display the results in a pretty table.

# create an empty vector with length of three

bfpaired <- numeric(3)

# calculate the bfs one by one enter them in vector

bfpaired[1] <- best3 %>% filter(runner %in% c("Usain Bolt", "Asafa Powell")) %>%

droplevels %>%

as.data.frame %>%

ttestBF(formula = timing ~ runner, data = .) %>%

extractBF %>%

.$bf

bfpaired[2] <- best3 %>% filter(runner %in% c("Usain Bolt", "Yohan Blake")) %>%

droplevels %>%

as.data.frame %>%

ttestBF(formula = timing ~ runner, data = .) %>%

extractBF %>%

.$bf

bfpaired[3] <- best3 %>% filter(runner %in% c("Asafa Powell", "Yohan Blake")) %>%

droplevels %>%

as.data.frame %>%

ttestBF(formula = timing ~ runner, data = .) %>%

extractBF %>%

.$bf

# round and append the values as a column named BF10

pairs\_compare$BF10 <- round(bfpaired,1)

# make a pretty table

kable(pairs\_compare, "html") %>%

kable\_styling(bootstrap\_options = c("striped", "hover", "condensed", "responsive")) %>%

add\_header\_above(c(" " = 4, "FWER" = 4, "FDR" = 2, "None" = 1, "Bayes" = 1))

|  | | | | **FWER** | | | | **FDR** | | **None** | **Bayes** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pair** | **meandif** | **stderr** | **tvalue** | **holm** | **hochberg** | **hommel** | **bonferroni** | **BH** | **BY** | **p.unadj** | **BF10** |
| Usain Bolt-Asafa Powell | -0.044 | 0.014 | -3.255 | 0.003 | 0.003 | 0.003 | 0.004 | 0.002 | 0.004 | 0.001 | 17.0 |
| Usain Bolt-Yohan Blake | -0.067 | 0.016 | -4.199 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 163.5 |
| Asafa Powell-Yohan Blake | -0.022 | 0.014 | -1.642 | 0.102 | 0.102 | 0.102 | 0.305 | 0.102 | 0.186 | 0.102 | 0.7 |

Bolt vs the other two runners yields no surprise. It quantifies the degree to which our evidence supports our hypothesized relationship but given the frequentist’s results we’re not surprised just happy to be able to make more probability based statements.

But Powell vs Blake is slightly disconcerting. The odds are 0.2924466 : 1 that Powell is faster than Blake? Now is the point at which we need to remember that one of the other benefits of a Bayesian approach is that it quantifies support for both the research hypothesis, which bayesians label **BF10**, as well as support for the null hypothesis called **BF01**! Therefore we can interpret that BF = .3 (rounded) as being the evidence provided by the data corresponds to odds of (1/0.2924466) **BF01** = 3:1 that the runners **ARE** equally fast.

### Even more nuance

But IMHO one of the most impressive things about the bayesian methodology is the ability to build complex models and to truly ask questions that are what you’re interested in and how the data support (or not) rather than simply rejecting a null hypothesis.

The next example benefited greatly from this post by the author of the BayesFactor package, Richard Morey. We’re not really interested in generic hypotheses that various runners aren’t equal. Oh no, as we sit in the bar we’re really interested in a very specific question. Our hypothesis is that if we were to arrange the dream race of the century and get Bolt, Powell and Blake on the 100m track that the finishing order would be Bolt first, Powell second and Blake third. The data when we view them visually seem to point us in that direction but it would be nice to get some sense of what the odds really are. We already have an empirical answer to one model stored in bf1 where the **BF10** = 144.1756106. Let’s start constructing a model that honors our actual question.

Our first step is to determine what we think our priors are before we have seen our data. Let’s take the view that we know nothing about male 100m runners and simply assume the runners have an equal chance of winning, I mean after all they are three of the fastest, if not **the** fastest, men alive. Three runners, how many different unique ways are there for them to finish (assuming ties are not possible)? The answer is quite simple but I’m going to code it out in r just so I have code for a future more complicated case. The gtools::permutations function allows us to map out all the possible unique finishing orders for our 3 runners. There are 6. Of the 6 only row #3 is what we are hypothesizing, none of the others match our research hypothesis. So our prior probabilities are 1/6 and we’ll store that in prior\_odds\_h1. Yes I know the code snippet below is overkill for the current problem and prior\_odds\_h1 <- 1 / 6 would have gotten me there faster without having to resort to loading the gtools package but I wanted to lay out a methodology for the future and possibly more complex cases.

possible\_finishes <- permutations(n = 3, r = 3, v = unique(levels(best3$runner)))

possible\_finishes

## [,1] [,2] [,3]

## [1,] "Asafa Powell" "Usain Bolt" "Yohan Blake"

## [2,] "Asafa Powell" "Yohan Blake" "Usain Bolt"

## [3,] "Usain Bolt" "Asafa Powell" "Yohan Blake"

## [4,] "Usain Bolt" "Yohan Blake" "Asafa Powell"

## [5,] "Yohan Blake" "Asafa Powell" "Usain Bolt"

## [6,] "Yohan Blake" "Usain Bolt" "Asafa Powell"

prior\_odds\_h1 <- 1 / nrow(possible\_finishes)

prior\_odds\_h1

## [1] 0.1666667

Okay we managed to get our “prior” with some very simple knowledge and math. To compute a posterior is a little trickier, and we’re going to need our friend the programmer to help us simulate some to get the answer. The bf1 object we created earlier has some useful information in it. A mathematical model formula for our posterior distribution. But it doesn’t allow us to directly generate the posterior probability we need to proceed. I’m not **even** going to try and explain the math but this post might help. For now in this little post I’m simply going to be satisfied with showing you how to use the posterior function in the BayesFactor package to come up with a reasonable and stable estimate.

Let’s set a random seed for reproducibility (on the chance you want your answer to match mine else you’re likely to get a slightly different answer every time because we are sampling). Use the posterior function and take a look at what we get. Focus on the column labeled **mu** (the grand mean for all 3 runners), and the columns labelled for each runner like **runner-Usain Bolt** which are how their mean time differed from the overall mean **mu**. A negative in this case indicates faster than the group mean. So we have 10,000 random simulations, and for each one we have a simulated draw from our data.

set.seed(1234)

samples <- posterior(bf1, iterations = 10000)

head(samples)

## Markov Chain Monte Carlo (MCMC) output:

## Start = 1

## End = 7

## Thinning interval = 1

## mu runner-Usain Bolt runner-Asafa Powell runner-Yohan Blake

## [1,] 9.860199 0.03941820 -0.0256338656 -0.01378433

## [2,] 9.937739 -0.02548752 -0.0123985442 0.03788606

## [3,] 9.925944 -0.02800456 0.0006957551 0.02730881

## [4,] 9.941116 -0.02558670 0.0051795110 0.02040719

## [5,] 9.936720 -0.02789285 0.0041579517 0.02373490

## [6,] 9.940669 -0.05242072 0.0070606025 0.04536012

## [7,] 9.937666 -0.03155442 -0.0044185658 0.03597299

## sig2 g\_runner

## [1,] 0.022299952 0.1726095

## [2,] 0.009297112 0.4232915

## [3,] 0.010127386 0.3618486

## [4,] 0.008350989 0.1429471

## [5,] 0.008973504 0.8421863

## [6,] 0.010552894 0.6322548

## [7,] 0.008594494 3.7694861

To get our estimate of the posterior probability then given our data and our model, all we have to do is count! What should we count? Well our hypothesis **H1** is that when we look at timing Bolt < Powell < Blake. So let’s write some r code to run through the 10,000 rows label each row TRUE if it supports our **H1** and FALSE if it doesn’t. Then we can use sum to count the “TRUES”. Once we have that count our posterior probability for our research hypothesis becomes the number that support divided by the total number of tries 10,000.

consistent\_with\_h1 <- (samples[, "runner-Usain Bolt"] < samples[, "runner-Asafa Powell"]) &

(samples[, "runner-Asafa Powell"] < samples[, "runner-Yohan Blake"])

head(consistent\_with\_h1, 20)

## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE

## [13] TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE

N\_consistent\_with\_h1 <- sum(consistent\_with\_h1)

N\_consistent\_with\_h1

## [1] 9332

posterior\_prob\_h1 <- N\_consistent\_with\_h1 / nrow(samples)

posterior\_prob\_h1

## [1] 0.9332

Now to calculate the Bayes Factor for this very specific and directional hypothesis all we need do is divide our posterior probability (our probability after seeing the data) by our prior prob which was our thing before we saw the data. The larger the **BF10** the more the evidence is persuading us that our model is “correct”.

bf\_\_h1 <- posterior\_prob\_h1 / prior\_odds\_h1

bf\_\_h1

## [1] 5.5992

Using Kass and Raftery (1995) guidelines we have “Positive evidence”

| **Bayes factor value** | **Interpretation** |
| --- | --- |
| 1 - 3 | Negligible evidence |
| 3 - 20 | Positive evidence |
| 20 -150 | Strong evidence |
| >150 | Very strong evidence |

It’s left to you to determine just how you feel about 5:1 odds. I suppose it would depend on what the stakes were and how convinced you needed to be.

#### Just a little farther

Before I end this post I want to accomplish two more things that I will address next post. The first is simply a way of encouraging you to make sure you understand this methodology. When you look at the distribution of the data plotted near the beginning of the post your eyes should “tell” you that while it is pretty clear Bolt is fastest the difference between Powell and Blake is quite small. Therefore the most likely reason we “only” get a BF of approximately 5:1 is the second part of our hypothesis Powell < Blake. So my challenge to you is to take the code and change it to make your research hypothesis that Bolt < Powell and that Bolt < Blake but make no assertion about Powell and Blake. **hint - pay attention to your prior too**. My next post I’ll address this and an even more complicated hypothesis.

Second, sometimes you want to compare nested models. If we think of the values stored in bf1 as our “full model” i.e., is any runner faster than any other runner in any direction we had a BF = ~7:1. Our more specific model gave us ~5:1. Bayes Factors are transitive and we can multiply them together to give us a sense of improvement very easily…

## extract just the BF from bf1 so that we can multiply it

bf\_full\_model <- as.vector(bf1)

bf\_full\_model

## runner

## 144.1756

## Use transitivity to compute a comparative Bayes factor

bf\_comparing <- bf\_\_h1 \* bf\_full\_model

bf\_comparing

## runner

## 807.2681

**Background**

So let’s load the necessary libraries.

library(rvest) # to ha"rvest" the web page

library(tidyverse) # using readr, dplyr, and purrr

library(ggstatsplot)

library(BayesFactor)

Next let’s duplicate Anindya’s earlier work and scrape [the Track and Field  
All-Time Performances webpage](http://www.alltime-athletics.com/m_100ok.htm) to  
get the data. One change I’m making is to remove n\_max = 3263 which is  
unnecessary and was preventing grabbing the newer race results from summer  
2019.

male\_100\_html <-

read\_html("http://www.alltime-athletics.com/m\_100ok.htm")

male\_100\_pres <- male\_100\_html %>%

html\_nodes(xpath = "//pre")

male\_100\_htext <- male\_100\_pres %>%

html\_text()

male\_100\_htext <- male\_100\_htext[[1]]

male\_100 <- read\_fwf(

male\_100\_htext,

skip = 1,

# n\_max = 3263, # n\_max removed to cpture newer races

col\_types = cols(.default = col\_character()),

col\_positions = fwf\_positions(

c(1, 16, 27, 35, 66, 74, 86, 93, 123),

c(15, 26, 34, 65, 73, 85, 92, 122, NA)

)

)

male\_100 <- male\_100 %>%

select(X2, X4) %>%

transmute(timing = X2, runner = X4) %>%

mutate(

timing = gsub("A", "", timing),

timing = as.numeric(timing)

)

# 3267 as of July 8, 2019

nrow(male\_100) # if you're cautious you can check against the webpage

## [1] 3267

Let’s focus on the **top 6** runners who have **more than 40 race** results  
recorded. We’ll make an effort throughout this post to capture the parameters we  
use and store them as variables and use the names. If you choose to replicate  
this post on your own you should be able to change the parameters below and see  
how the results vary based upon your choices (for example the top 5 or 10  
runners or more than 30 races).

numbraces <- 40

howmanyrunners <- 6

Having made our selections let’s use a series a dplyr commands piped %>%  
together to create a character vector called orderbymean which contains the  
names of the 6 runners who meet our criteria. We can use this  
vector to filter our dataframe down to just the 6 we want with  
a filter(runner %in% orderbymean) statement as well as force the factor levels  
of runner to be in mean order with factor(male\_100$runner, levels = orderbymean).

orderbymean <- male\_100 %>%

group\_by(runner) %>%

summarise(avgtime = mean(timing), races = n()) %>%

arrange(avgtime) %>%

filter(races >= numbraces) %>%

top\_n(-howmanyrunners, avgtime) %>%

pull(runner) %>%

as.character()

orderbymean

## [1] "Usain Bolt" "Asafa Powell" "Tyson Gay" "Justin Gatlin" "Yohan Blake" "Maurice Greene"

male\_100 <- male\_100 %>%

filter(runner %in% orderbymean) %>%

mutate\_if(is.character, as.factor) %>%

droplevels()

male\_100$runner <-

factor(

male\_100$runner,

levels = orderbymean

)

glimpse(male\_100)

## Observations: 530

## Variables: 2

## $ timing 9.58, 9.63, 9.69, 9.69, 9.69, 9.71, 9.72, 9.72, 9.74, 9.74, 9.75, 9.75, 9.75, 9.75, 9.76, 9.…

## $ runner Usain Bolt, Usain Bolt, Usain Bolt, Tyson Gay, Yohan Blake, Tyson Gay, Usain Bolt, Asafa Pow…

levels(male\_100$runner)

## [1] "Usain Bolt" "Asafa Powell" "Tyson Gay" "Justin Gatlin" "Yohan Blake" "Maurice Greene"

Okay we now have the 530 relevant times for the 6  
runners we’re focusing on. The focus of this post is to build  
the bayesian equivalent of a frequentist’s pairwise comparisons test across all  
the unique runner pairings pairwise.t.test(x = male\_100$timing, g = male\_100$runner, p.adjust.method = "holm"). As I mentioned this  
would be the logical next step after conducting a oneway ANOVA of timing ~ runner or it’s bayesian equivalent BayesFactor::anovaBF(timing ~ runner, male\_100)

##

## Pairwise comparisons using t tests with pooled SD

##

## data: male\_100$timing and male\_100$runner

##

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake

## Asafa Powell 0.0460 - - - -

## Tyson Gay 0.0250 1.0000 - - -

## Justin Gatlin 0.0024 1.0000 1.0000 - -

## Yohan Blake 0.0052 1.0000 1.0000 1.0000 -

## Maurice Greene 6.9e-05 0.1649 1.0000 1.0000 1.0000

##

## P value adjustment method: holm

Our Bayes equivalent to this matrix won’t report *“p values”* but rather the  
Bayes Factor associated with the pairing. Unlike the frequentist’s  
**“reject/don’t reject”** criteria, the BF we report will indicate what the data  
provide as odds that our hypothesis is correct. We’ll build it methodically and  
with an eye towards code that is easily reused in the future.

So, quick quiz, how many unique pair combinations are there for our 6  
runners? The order of the pair doesn’t matter at this juncture  
we’re looking for the number of possible head to head races among these 6  
runners. If you’re like me you don’t know the answer to that off  
of the top of your head and it can be tedious figuring it out, so let’s let the  
computer always calculate it for us, and tell us. For this case it’s  
15.  
Then we can use combn to take the 6 names and show us what those pairings are  
e.g., *“Usain Bolt, Asafa Powell”*. The result is a matrix of 15 columns  
(all the pairings) and two rows (the names of the runners for the pairings).  
**Just to make it easier to see I’ve added a t() so the display is vertical  
and you can see the pairs.**

numberofpairings <- factorial(howmanyrunners) /

(factorial(2) \* factorial(howmanyrunners - 2))

numberofpairings

## [1] 15

t(combn(orderbymean, 2))

## [,1] [,2]

## [1,] "Usain Bolt" "Asafa Powell"

## [2,] "Usain Bolt" "Tyson Gay"

## [3,] "Usain Bolt" "Justin Gatlin"

## [4,] "Usain Bolt" "Yohan Blake"

## [5,] "Usain Bolt" "Maurice Greene"

## [6,] "Asafa Powell" "Tyson Gay"

## [7,] "Asafa Powell" "Justin Gatlin"

## [8,] "Asafa Powell" "Yohan Blake"

## [9,] "Asafa Powell" "Maurice Greene"

## [10,] "Tyson Gay" "Justin Gatlin"

## [11,] "Tyson Gay" "Yohan Blake"

## [12,] "Tyson Gay" "Maurice Greene"

## [13,] "Justin Gatlin" "Yohan Blake"

## [14,] "Justin Gatlin" "Maurice Greene"

## [15,] "Yohan Blake" "Maurice Greene"

**Purrring right along**

Now we can grab the output from combn and create two separate vectors,  
runner1 and runner2. We’ll take those vectors and create a series of purrr  
statements using pipes.

* The first is a map2 which takes runner1 and runner2 and creates a list  
  of 15 dataframes, one for each pairing. The anonymous function(a, b) is  
  simply an organized way of working our way through the 15 pairings. Filtering  
  and dropping levels and explicitly converting to a dataframe because  
  BayesFactor::ttestBF will generate a warning if you pass it a tibble.
* Next we purrr::map the list we just created (.x = .) and call the  
  ttestBF function. For each item in the list of 15 dataframes (one for each  
  runner pairing) it runs with formula = timing ~ runner, the dataframe we  
  passed data = ., and in this case we have deliberately specified a  
  directional hypothesis nullInterval = c(-Inf, 0) (it’s **“-Inf”** *because  
  the faster runner has a smaller timing*).
* for a more complete explanation of directional hypothesis testing.
* We started with two vectors runner1 and runner2 after map2 we had a list  
  of 15 dataframes. Now after the first map pipe we have a list of 15 Bayes  
  Factor objects (that’s what ttestBF generates). We’ll immediately pipe (%>%)  
  that list (.x = .) into another map where we’ll invoke the extractBF  
  function. extractBF produces a dataframe, in this case with 2 rows, one with  
  the rowname “Alt., r=0.707 -Inf r=0.707 !(-Infwhich we don’t need.
* Now we have a list of 15 dataframes, this time containing the results of our  
  ttestBF. We want the row in each of them that is named “Alt., r=0.707  
  -Inf stored. So what we want is a list of 15 real numbers. So this time we’ll use  
  map\_dbl to let it know we want a list of 15 numbers, map\_dbl(.x = ., ~ .["Alt., r=0.707 -Inf.
* The final pipe simply does some trivial rounding.

runner1 <- combn(orderbymean, 2)[1, ]

runner1

## [1] "Usain Bolt" "Usain Bolt" "Usain Bolt" "Usain Bolt" "Usain Bolt" "Asafa Powell"

## [7] "Asafa Powell" "Asafa Powell" "Asafa Powell" "Tyson Gay" "Tyson Gay" "Tyson Gay"

## [13] "Justin Gatlin" "Justin Gatlin" "Yohan Blake"

runner2 <- combn(orderbymean, 2)[2, ]

runner2

## [1] "Asafa Powell" "Tyson Gay" "Justin Gatlin" "Yohan Blake" "Maurice Greene" "Tyson Gay"

## [7] "Justin Gatlin" "Yohan Blake" "Maurice Greene" "Justin Gatlin" "Yohan Blake" "Maurice Greene"

## [13] "Yohan Blake" "Maurice Greene" "Maurice Greene"

bfresults <- map2(

runner1,

runner2,

function(a, b)

male\_100 %>%

filter(runner %in% c(a, b)) %>%

droplevels() %>%

as.data.frame()

) %>%

map(.x = ., ~ ttestBF(

formula = timing ~ runner,

data = .,

nullInterval = c(-Inf, 0)

)) %>%

map(.x = ., ~ extractBF(x = .)) %>%

map\_dbl(.x = ., ~ .["Alt., r=0.707 -Inf%

round(., digits = 4)

bfresults

## [1] 10.2865 9.4539 59.3688 36.4792 2709.0974 0.3217 0.5782 0.7394 8.5068 0.2271

## [11] 0.2902 0.8574 0.2033 0.5368 0.4184

To some, the complex set of steps that leads to bfresults may look daunting.  
I’d be a liar if I tried to say I wrote all those lines in one pass and got  
everything right. My suggestion is that as you build the pipeline you work step  
by step producing intermediate objects. Once you get the individual steps  
correct it’s trivial to join them using %>% and .x = ..

Now that we have our 15 bayes factors for each of the 15 pairings of runners we  
should probably join them together into one neat dataframe resultsdf that lays  
everything out for us. Based on the data available we would read line #5 as the  
odds are 2709:1 that Usain is faster than Maurice.

resultsdf <-

data.frame(

Runner1 = runner1,

Runner2 = runner2,

oddsfaster = bfresults

)

resultsdf

## Runner1 Runner2 oddsfaster

## 1 Usain Bolt Asafa Powell 10.2865

## 2 Usain Bolt Tyson Gay 9.4539

## 3 Usain Bolt Justin Gatlin 59.3688

## 4 Usain Bolt Yohan Blake 36.4792

## 5 Usain Bolt Maurice Greene 2709.0974

## 6 Asafa Powell Tyson Gay 0.3217

## 7 Asafa Powell Justin Gatlin 0.5782

## 8 Asafa Powell Yohan Blake 0.7394

## 9 Asafa Powell Maurice Greene 8.5068

## 10 Tyson Gay Justin Gatlin 0.2271

## 11 Tyson Gay Yohan Blake 0.2902

## 12 Tyson Gay Maurice Greene 0.8574

## 13 Justin Gatlin Yohan Blake 0.2033

## 14 Justin Gatlin Maurice Greene 0.5368

## 15 Yohan Blake Maurice Greene 0.4184

**The Matrix reloaded (still waiting for #4)**

Now that we have our resultsdf we can continue about the business of comparing  
the frequentist results of paired t-tests with their bayesian counterparts.  
Imagine that we have just completed a Oneway ANOVA of timing ~ runner (I’ll  
show the results in a bit). Given significant results of the omnibuds **F test**  
our next step is likely to run all the pairwise comparisons with some sort of  
correction for multiple comparisons like pairwise.t.test.

The results are almost always given as a matrix often without repeating one of  
the diagonals. The results tell us that we can reject the null hypothesis that  
the runners have the same time for Usain Bolt versus all the other competitors.  
But it doesn’t allow us to make any statements about how different (**despite  
the temptation inherent in the very different p values**). It supplies almost no  
information about the other pairings, just that we can not reject the null.

pairwise.t.test(

x = male\_100$timing,

g = male\_100$runner,

p.adjust.method = "holm"

)

##

## Pairwise comparisons using t tests with pooled SD

##

## data: male\_100$timing and male\_100$runner

##

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake

## Asafa Powell 0.0460 - - - -

## Tyson Gay 0.0250 1.0000 - - -

## Justin Gatlin 0.0024 1.0000 1.0000 - -

## Yohan Blake 0.0052 1.0000 1.0000 1.0000 -

## Maurice Greene 6.9e-05 0.1649 1.0000 1.0000 1.0000

##

## P value adjustment method: holm

Another typical way of displaying the information is graphically as demonstrated  
here by ggstatsplot::ggbetweenstats.

ggbetweenstats(

data = male\_100,

x = runner,

y = timing,

type = "p",

var.equal = TRUE,

pairwise.comparisons = TRUE,

pairwise.display = "all",

partial = FALSE,

effsize.type = "unbiased",

sort = "ascending",

point.jitter.height = 0,

messages = FALSE

)

Let’s see if we can’t at least produce a similar matrix to what  
pairwise.t.test yields. I’d like us to be able to do a sort of side by side  
comparison of the frequentists versus bayesian results.

Step by step the process we’ll follow is:

1. Use diag to create a matrix with ones in the diagonal we’ll set the size to  
   howmanyrunners
2. Grab the runners names from orderbymean and populate  
   the rownames and colnames
3. Use combn again this time populating it with numbers (one & two) rather  
   than the runners names
4. Feed those vectors into a for loop to populate the bfmatrix with the data  
   from resultsdf
5. To be consistent with pairwise.t.test remove the first row bfmatrix[-1, ]  
   and the last column bfmatrix[, -howmanyrunners]
6. Finally populate the upper triangle part of the matrix with NA  
   bfmatrix[upper.tri(bfmatrix)] <- NA

bfmatrix <- diag(nrow = howmanyrunners)

rownames(bfmatrix) <- orderbymean

colnames(bfmatrix) <- orderbymean

bfmatrix

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake Maurice Greene

## Usain Bolt 1 0 0 0 0 0

## Asafa Powell 0 1 0 0 0 0

## Tyson Gay 0 0 1 0 0 0

## Justin Gatlin 0 0 0 1 0 0

## Yohan Blake 0 0 0 0 1 0

## Maurice Greene 0 0 0 0 0 1

combn(howmanyrunners, 2)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]

## [1,] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5

## [2,] 2 3 4 5 6 3 4 5 6 4 5 6 5 6 6

one <- combn(howmanyrunners, 2)[2, ]

two <- combn(howmanyrunners, 2)[1, ]

for (i in 1:numberofpairings) {

bfmatrix[one[i], two[i]] <- resultsdf[i, 3] # row i, column 3 which is the BF value

}

bfmatrix

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake Maurice Greene

## Usain Bolt 1.0000 0.0000 0.0000 0.0000 0.0000 0

## Asafa Powell 10.2865 1.0000 0.0000 0.0000 0.0000 0

## Tyson Gay 9.4539 0.3217 1.0000 0.0000 0.0000 0

## Justin Gatlin 59.3688 0.5782 0.2271 1.0000 0.0000 0

## Yohan Blake 36.4792 0.7394 0.2902 0.2033 1.0000 0

## Maurice Greene 2709.0974 8.5068 0.8574 0.5368 0.4184 1

bfmatrix <- bfmatrix[-1, ]

bfmatrix <- bfmatrix[, -howmanyrunners]

bfmatrix

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake

## Asafa Powell 10.2865 1.0000 0.0000 0.0000 0.0000

## Tyson Gay 9.4539 0.3217 1.0000 0.0000 0.0000

## Justin Gatlin 59.3688 0.5782 0.2271 1.0000 0.0000

## Yohan Blake 36.4792 0.7394 0.2902 0.2033 1.0000

## Maurice Greene 2709.0974 8.5068 0.8574 0.5368 0.4184

bfmatrix[upper.tri(bfmatrix)] <- NA

bfmatrix

## Usain Bolt Asafa Powell Tyson Gay Justin Gatlin Yohan Blake

## Asafa Powell 10.2865 NA NA NA NA

## Tyson Gay 9.4539 0.3217 NA NA NA

## Justin Gatlin 59.3688 0.5782 0.2271 NA NA

## Yohan Blake 36.4792 0.7394 0.2902 0.2033 NA

## Maurice Greene 2709.0974 8.5068 0.8574 0.5368 0.4184

Success! bfmatrix now looks a lot like the results of pairwise.t.test. But  
we can do better. Looking at the object produced by pairwise.t.test using  
str() we can see it is a list with 4 items. $method contains a simple text  
string explaining what it is. $data.name is another text string that tells us  
where the data came from. $p.value contains the actual p values and finally  
$p.adjust.method contains which method of p adjustment (*holm*) we used. Since  
there is no analog *“adjustment method”* for a bayes factor we can ignore it.  
Let’s make our own list called bfpairs that mimics that structure.

str(pairwise.t.test(

x = male\_100$timing,

g = male\_100$runner,

p.adjust.method = "holm"

))

## List of 4

## $ method : chr "t tests with pooled SD"

## $ data.name : chr "male\_100$timing and male\_100$runner"

## $ p.value : num [1:5, 1:5] 4.60e-02 2.50e-02 2.44e-03 5.15e-03 6.89e-05 ...

## ..- attr(\*, "dimnames")=List of 2

## .. ..$ : chr [1:5] "Asafa Powell" "Tyson Gay" "Justin Gatlin" "Yohan Blake" ...

## .. ..$ : chr [1:5] "Usain Bolt" "Asafa Powell" "Tyson Gay" "Justin Gatlin" ...

## $ p.adjust.method: chr "holm"

## - attr(\*, "class")= chr "pairwise.htest"

bfpairs <- list(

method = " r = 0.707 Alt Hyp = -Inf

## List of 3

## $ method : chr " r = 0.707 Alt Hyp = -Inf

What we’d like to do is mimic the print method for pairwise.t.test to include  
little things like substituting in an *em dash* instead of the NAs. To do that  
we need to peek at the print method with getAnywhere(print.pairwise.htest).  
getAnywhere() is a life saver if you want to be able to inspect a function.

getAnywhere(print.pairwise.htest)

## A single object matching 'print.pairwise.htest' was found

## It was found in the following places

## registered S3 method for print from namespace stats

## namespace:stats

## with value

##

## function (x, digits = max(1L, getOption("digits") - 5L), ...)

## {

## cat("\n\tPairwise comparisons using", x$method, "\n\n")

## cat("data: ", x$data.name, "\n\n")

## pp <- format.pval(x$p.value, digits = digits, na.form = "-")

## attributes(pp) <- attributes(x$p.value)

## print(pp, quote = FALSE, ...)

## cat("\nP value adjustment method:", x$p.adjust.method, "\n")

## invisible(x)

## }

##

##

Hmmmmm. Okay so we pass it a list object (the bfpairs we just created) and it  
takes the sub components and puts them into the right places on the screen.  
Spoiler alert, I won’t go into all the details but suffice it to say that  
format.pval() is problematic for us.  
It does a very nice job working with p values but p values have a different set  
of characteristics than bayes factors.

Rather than modify format.pval I simply decided to use the generic format  
function instead. That way the end user can specify all sorts of parameters like  
the number of digits, the symbol to replace NA, and the justification etc..

Here’s what I came up with after a little bit of work. Hopefully you’ll agree it  
does a reasonably good job of replicating the functionality of  
print.pairwise.htest?

print.pairwise.bftest <- function(x,

digits = 2,

nsmall = 0,

width = 9,

justify = "right",

scientific = FALSE,

nareplace = "-") {

cat("\nPairwise comparisons of bayes factors with", x$method, "\n\n")

cat("data: ", x$data.name, "\n\n")

pp <- format(x$p.value,

digits = digits,

nsmall = nsmall,

width = width,

justify = justify,

scientific = scientific

)

pp <- gsub("NA", nareplace, pp)

print(pp, quote = FALSE)

cat("\n\nAnalyzed using BayesFactor::ttestBF\n")

invisible(x)

}

print.pairwise.bftest(bfpairs, digits = 1)

##

## Pairwise comparisons of bayes factors with r = 0.707 Alt Hyp = -Inf

Notice that bayes factors aren’t shockingly dissimilar than the conclusions you  
would draw from a frequentist’s perspective. I still think they are a better  
choice because you can talk about odds and probabilities cleanly without falling  
into the frequentist “traps” surrounding what rejection of the null hypothesis  
is. With our “new” perspective we are safe in making statements that out data  
strongly support some of the pairwise differences (odds of 2709 to 1 are pretty  
convincing) and in other cases we can now quantify that odds are it’s “anyone’s  
race.”

**Play it again Sam**

As I wrote this post I wanted to ensure that I could run the analysis on a  
different set of runners with minimal effort. What follows is the code minus all  
of the intermediate printing and explanation. The difference is this time  
we’ll look at the top 7 fastest sprinters and widen our analysis to anyone with  
at least 20 races.

male\_100\_html <-

read\_html("http://www.alltime-athletics.com/m\_100ok.htm")

male\_100\_pres <- male\_100\_html %>%

html\_nodes(xpath = "//pre")

male\_100\_htext <- male\_100\_pres %>%

html\_text()

male\_100\_htext <- male\_100\_htext[[1]]

male\_100 <- read\_fwf(

male\_100\_htext,

skip = 1,

col\_types = cols(.default = col\_character()),

col\_positions = fwf\_positions(

c(1, 16, 27, 35, 66, 74, 86, 93, 123),

c(15, 26, 34, 65, 73, 85, 92, 122, NA)

)

)

male\_100 <- male\_100 %>%

select(X2, X4) %>%

transmute(timing = X2, runner = X4) %>%

mutate(

timing = gsub("A", "", timing),

timing = as.numeric(timing)

)

numbraces <- 20

howmanyrunners <- 7

orderbymean <- male\_100 %>%

group\_by(runner) %>%

summarise(avgtime = mean(timing), races = n()) %>%

arrange(avgtime) %>%

filter(races >= numbraces) %>%

top\_n(-howmanyrunners, avgtime) %>%

pull(runner) %>%

as.character()

male\_100 <- male\_100 %>%

filter(runner %in% orderbymean) %>%

mutate\_if(is.character, as.factor) %>%

droplevels()

male\_100$runner <-

factor(

male\_100$runner,

levels = orderbymean

)

numberofpairings <- factorial(howmanyrunners) /

(factorial(2) \* factorial(howmanyrunners - 2))

runner1 <- combn(orderbymean, 2)[1, ]

runner2 <- combn(orderbymean, 2)[2, ]

bfresults <- map2(

runner1,

runner2,

function(a, b)

male\_100 %>%

filter(runner %in% c(a, b)) %>%

droplevels() %>%

as.data.frame()

) %>%

map(.x = ., ~ ttestBF(

formula = timing ~ runner,

data = .,

nullInterval = c(-Inf, 0)

)) %>%

map(.x = ., ~ extractBF(x = .)) %>%

map\_dbl(.x = ., ~ .["Alt., r=0.707 -Inf%

round(., digits = 4)

resultsdf <-

data.frame(

Runner1 = runner1,

Runner2 = runner2,

oddsfaster = bfresults

)

bfmatrix <- diag(nrow = howmanyrunners)

rownames(bfmatrix) <- orderbymean

colnames(bfmatrix) <- orderbymean

one <- combn(howmanyrunners, 2)[2, ]

two <- combn(howmanyrunners, 2)[1, ]

for (i in 1:numberofpairings) {

bfmatrix[one[i], two[i]] <- resultsdf[i, 3]

}

bfmatrix <- bfmatrix[-1, ]

bfmatrix <- bfmatrix[, -howmanyrunners]

bfmatrix[upper.tri(bfmatrix)] <- NA

bfpairs <- list(

method = " r = 0.707 Alt Hyp = -Inf

##

## Pairwise comparisons using t tests with pooled SD

##

## data: male\_100$timing and male\_100$runner

##

## Usain Bolt Asafa Powell Tyson Gay Christian Coleman Justin Gatlin Yohan Blake

## Asafa Powell 0.0682 - - - - -

## Tyson Gay 0.0358 1.0000 - - - -

## Christian Coleman 0.2500 1.0000 1.0000 - - -

## Justin Gatlin 0.0033 1.0000 1.0000 1.0000 - -

## Yohan Blake 0.0071 1.0000 1.0000 1.0000 1.0000 -

## Maurice Greene 8.7e-05 0.2500 1.0000 1.0000 1.0000 1.0000

##

## P value adjustment method: holm

print.pairwise.bftest(bfpairs,

digits = 3,

scientific = TRUE,

nareplace = ".")

##

## Pairwise comparisons of bayes factors with r = 0.707 Alt Hyp = -Inf

And voila! Based on our new criteria of the fastest 7 runners  
with at least 20 races Christian Coleman has been added to the  
matrix. His mean timings place him square in the middle of the pack between  
Tyson Gay and Justin Gatlin. But notice the BF comparing him to Usain Bolt is  
only about 3 which is smaller than Tyson Gay  
9.5 and Justin Gatlin 59.4 or any of  
the other runners. This is likely because the BF always adjusts based upon the  
amount of evidence available and we only have  
23  
races of data available for Christian.

Remember that one of the nice features of bayesian methodology is that we can  
quantify support for both the hypothesis we have as well as it’s converse (what  
a frequentist would call the null hypothesis). So our hypothesis is that Justin  
Gatlin is faster than Yohan Blake but the bayes factor 2.03e-01 (.203) says that  
the evidence from the data is that the odds are 1 / 2.03e-01 or about 5:1 that  
Justin is **NOT** faster than Yohan. That’s a statement that can not be made  
when using frequentist methods.

**Done**

I’ve really enjoyed this series of posts. I am always open to comments,  
corrections and suggestions. Feel free to leave a comment in disqus or send me  
an email.