# The Very Basics

If you have some background in statistics the next section may be a little boring for you but bear with me I promise an opportunity to delve more deeply later and revisit more complex issues like the the differences between frequentist and bayesian methods later.

I’ve explored this topic before

but there’s plenty of new stuff below. Let’s just get the basics under our belts.

If you’ve ever taken an introductory course in probability and statistics you’ve likely been exposed to the statistician’s love of coin flips. This is another instance where at it’s simplest comparing our vaccine trials to coin flips can be instructive. Although overall the study is a

randomized controlled trial

(actually several of them) on a very large scale in many ways our understanding of these first 53 cases can be modeled as a set of coin flips. For each of the

53 people who are confirmed to have contracted covid-19 they either got the vaccine or a placebo (heads or tails).

R let’s us quickly get a sense of what the probability is that of the 53 people in the trial who contracted covid-19 what if only the minimum of 5 were given the actual vaccine and not the placebo? For those of you who don’t like scientific notation that’s thirty-one billion, eight hundred fifty-nine million,

nine hundred thousand probability against you doing that randomly flipping coins.

got\_covid <- 53

vaccinated\_got\_covid <- 5

placebo\_got\_covid <- 53 - vaccinated\_got\_covid dbinom(vaccinated\_got\_covid, got\_covid, .5)

## [1] 3.18599e-10

vaccinated\_got\_covid <- 19

placebo\_got\_covid <- 53 - vaccinated\_got\_covid dbinom(vaccinated\_got\_covid, got\_covid, .5)

## [1] 0.01321525

Even if 19 people who received the true vaccine got covid the probability is still less than 2% that the vaccine versus placebo doesn’t matter that it’s a 50/50 chance.

Let’s talk about effectiveness. We want our vaccine to be at least 50% effective. We can operationalize that most simply by

\[\frac{placebo~cases - vaccinated~cases}{placebo~cases}\]

So for our current example where 19 of the 53 cases had received the vaccine our effectiveness is

\[\frac{placebo~cases - vaccinated~cases}{placebo~cases} = (34-19)/34 = 0.4411765\] which is very close to what we need. Let’s display it graphically for the range

between 5 and 26 vaccinated cases and dispense with one other pesky issue. So far we have been ignoring the fact that there aren’t 53 people involved in the

trial, there are 30,000. We don’t really want to lose sight of this even though it makes very little difference to our math in most cases. Let’s calculate

effectiveness both with and without 30,000 in the denominator and even track

rounding error in our little tibble.

library(dplyr) library(ggplot2) library(kableExtra) theme\_set(theme\_bw())

effectiveness <- tibble::tibble(vaccinated = 5:26) %>% mutate(placebo = 53 - vaccinated,

effectiveness = (placebo - vaccinated) / placebo \* 100, placeborate = placebo / (15000 - placebo), vaccinatedrate = vaccinated / (15000 - vaccinated), pctratedifference = (placeborate - vaccinatedrate) /

placeborate \* 100,

rounding = effectiveness - pctratedifference)

effectiveness %>%

kbl(digits = c(0,0,2,4,4,2,2),

caption = "Effectiveness as a function of positive COVID cases")

%>%

kable\_minimal(full\_width = FALSE, position = "left") %>%

add\_header\_above(c("Cases (53)" = 2, " " = 1, "Infection Rate" = 2, "% difference in rate" = 1, " " = 1))

Table 1: Effectiveness as a function of positive COVID cases

## Cases (53) Infection Rate % difference in rate

**vaccinated placebo effectiveness placeborate vaccinatedrate pctratedifference rounding**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 48 | 89.58 | 0.0032 | 0.0003 | 89.61 | -0.03 |
| 6 | 47 | 87.23 | 0.0031 | 0.0004 | 87.27 | -0.03 |
| 7 | 46 | 84.78 | 0.0031 | 0.0005 | 84.82 | -0.04 |
| 8 | 45 | 82.22 | 0.0030 | 0.0005 | 82.27 | -0.04 |
| 9 | 44 | 79.55 | 0.0029 | 0.0006 | 79.59 | -0.05 |
| 10 | 43 | 76.74 | 0.0029 | 0.0007 | 76.80 | -0.05 |
| 11 | 42 | 73.81 | 0.0028 | 0.0007 | 73.86 | -0.05 |
| 12 | 41 | 70.73 | 0.0027 | 0.0008 | 70.79 | -0.06 |
| 13 | 40 | 67.50 | 0.0027 | 0.0009 | 67.56 | -0.06 |
| 14 | 39 | 64.10 | 0.0026 | 0.0009 | 64.16 | -0.06 |
| 15 | 38 | 60.53 | 0.0025 | 0.0010 | 60.59 | -0.06 |
| 16 | 37 | 56.76 | 0.0025 | 0.0011 | 56.82 | -0.06 |
| 17 | 36 | 52.78 | 0.0024 | 0.0011 | 52.84 | -0.06 |
| 18 | 35 | 48.57 | 0.0023 | 0.0012 | 48.63 | -0.06 |
| 19 | 34 | 44.12 | 0.0023 | 0.0013 | 44.17 | -0.06 |
| 20 | 33 | 39.39 | 0.0022 | 0.0013 | 39.45 | -0.05 |
| 21 | 32 | 34.38 | 0.0021 | 0.0014 | 34.42 | -0.05 |
| 22 | 31 | 29.03 | 0.0021 | 0.0015 | 29.07 | -0.04 |

## Cases (53) Infection Rate % difference in rate

**vaccinated placebo effectiveness placeborate vaccinatedrate pctratedifference rounding**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 23 | 30 | 23.33 | 0.0020 | 0.0015 | 23.37 | -0.04 |
| 24 | 29 | 17.24 | 0.0019 | 0.0016 | 17.27 | -0.03 |
| 25 | 28 | 10.71 | 0.0019 | 0.0017 | 10.73 | -0.02 |
| 26 | 27 | 3.70 | 0.0018 | 0.0017 | 3.71 | -0.01 |

effectiveness %>%

ggplot(aes(x = vaccinated)) + geom\_line(aes(y = effectiveness)) +

geom\_line(aes(y = pctratedifference), color = "green") + geom\_hline(aes(yintercept = 50)) +

scale\_y\_continuous(labels = scales::label\_percent(scale = 1),

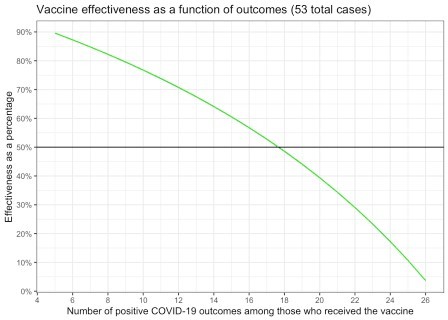
breaks = seq.int(0, 100, 10)) +

scale\_x\_continuous(breaks = seq.int(2, 26, 2)) +

xlab("Number of positive COVID-19 outcomes among those who received the vaccine") +

ylab("Effectiveness as a percentage") +

ggtitle("Vaccine effectiveness as a function of outcomes (53 total cases)")



mean(effectiveness$rounding) # that's .04% not 4.0%

## [1] -0.04559179

So in the best possible scenario (remember we require that at least 5 people who got the vaccine contract COVID before we can run the numbers) our effectiveness

is ~90%. We need the number of people who received the vaccine and still contracted COVID to be 17 or less.

# Testing our confidence as a frequentist

Now we know what we have to do to convince ourselves about effectiveness let’s address what we can do to be confident that our results are not a fluke. We

know that low probability events occur. I don’t want to repeat myself so if you want a little background on frequentist methodology

Having said all that, frequentist methods are certainly the most prevalent and likely to be applied to the warpspeed data so let’s see what we come up with. Let’s start with the simplest possible test we could use. Let’s build a simple matrix (table) of our results and call it dat. We’ll assume the 30,000 participants are equally divided between receiving the vaccine and the placebo. We’ll pretend 19 folks who got the real vaccine later developed COVID and 34 did not. We’ll grab just the first column dat[,1] and put that in an

object called covid\_gof. Our hypothesis is that the vaccine matters that it helps prevent being infected. In NHST terms our null hypothesis is that there is no difference in the number of people who will get infected. In essence it might as well be a coin toss, 50/50, equal numbers of people will get infected whether they received the vaccine or the placebo. We can use the \(\chi^2\) goodness of fit test. For clarity we’ll remove the continuity correction and overtly specify 50/50 odds. Since the default chisq.test

test results are very terse we’ll use lsr::goodnessOfFitTest on the same data (expressed as vector that is a factor) to make it clear what we’re

doing.

dat <- matrix(c(34, 15000 - 34, 19, 15000 - 19),

nrow = 2, byrow = TRUE)

rownames(dat) <- c("Placebo", "Vaccine") colnames(dat) <- c("COVID", "No COVID") dat

## COVID No COVID

## Placebo 34 14966

## Vaccine 19 14981

covid\_gof <- dat[,1]

chisq.test(covid\_gof,

correct = FALSE, p = c(.5, .5))

##

## Chi-squared test for given probabilities ##

## data: covid\_gof

## X-squared = 4.2453, df = 1, p-value = 0.03936

outcomes <- factor(c(rep("Vaccine", 19), rep("Placebo", 34))) lsr::goodnessOfFitTest(outcomes)

##

## Chi-square test against specified probabilities ##

## Data variable: outcomes ##

## Hypotheses:

## null: true probabilities are as specified

## alternative: true probabilities differ from those specified ##

## Descriptives:

## observed freq. expected freq. specified prob. ## Placebo 34 26.5 0.5

## Vaccine 19 26.5 0.5

##

|  |  |  |
| --- | --- | --- |
| ## | Test results: |  |
| ## | X-squared statistic: | 4.245 |
| ##  ## | degrees of freedom:  p-value: 0.039 | 1 |

Either way \(\chi^2\) = 4.245283

and p = 0.0393595. Therefore we reject

the null and can have some “confidence” that our results aren’t simply a fluke.

That seems a little too simple and seems to ignore the fact that we actually have data for not 53 people but 30,000 people. So let’s make things more complex. There are actually many variants of a \(\chi^2\) test. Instead of goodness of fit let’s use \(\chi^2\) test of independence or association using all four cells of our little matrix. Here our null hypothesis is that

our variables are independent of one another, that whether you get COVID has no association to whether you received the vaccine or the placebo. A subtle distinction perhaps but worth it if for no other reason than to exploit the additional data. To make it even more fun there are numerous functions in

r to run the test. From base r, where there are at least two variants

chisq.test and prop.test

to specialized packages like epiR

that focus on epidemiology.

Notice that they all express the same basic notions

\(\chi^2\) = 4.2527963

and p = 0.0391858, even though they present an array of additional information.

chisq.test(dat, correct = FALSE) ##

## Pearson's Chi-squared test ##

## data: dat

## X-squared = 4.2528, df = 1, p-value = 0.03919 prop.test(dat, correct = FALSE)

##

## 2-sample test for equality of proportions without continuity ## correction

##

## data: dat

## X-squared = 4.2528, df = 1, p-value = 0.03919 ## alternative hypothesis: two.sided

## 95 percent confidence interval: ## 0.0000496578 0.0019503422

## sample estimates:

## prop 1 prop 2

## 0.002266667 0.001266667

epiR::epi.2by2(dat = as.table(dat), method = "cross.sectional",

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| conf.level = 0.95, units = 1, outcome = "as.columns") | | | | | | | |
| ## Outcome + | | | | Outcome - | Total | Prevalence \* | |
| Odds | | | |  |  |  | |
| ## Exposed + 34 | | | | 14966 | 15000 | 0.00227 | |
| 0.00227 | | | |  |  |  | |
| ## Exposed - 19 | | | | 14981 | 15000 | 0.00127 | |
| 0.00127 | | | |  |  |  | |
| ## Total 53 | | | | 29947 | 30000 | 0.00177 | |
| 0.00177 | | | |  |  |  | |
| ## | | | |  |  |  | |
| ## Point estimates and 95% | | | | CIs: |  |  | |
| ## | - | | | | | | |
| ## | Prevalence ratio | |  | 1.79 | | (1.02, | 3.14) |
| ## | Odds ratio | |  | 1.79 | | (1.02, | 3.14) |
| ## | Attrib prevalence | | \* | 0.00 | | (0.00, | 0.00) |
| ## Attrib prevalence in population \* 0.00 (-0.00, 0.00) | | | | | | | |
| ## | Attrib | fraction in exposed (%) | | | 44.12 | (2.08, 68.11) | |
| ## | Attrib | fraction in population (%) | | | 28.30 | (-2.75, 49.97) | |
| ## - | | | | | | | |

## Test that OR = 1: chi2(1) = 4.253 Pr>chi2 = 0.04 ## Wald confidence limits

## CI: confidence interval

## \* Outcomes per population unit

Notice that epiR::epi.2by2 provides confirmation of several important pieces of data we generated earlier in our very first table results. If you consult the row for 19 & 34, **prevalence** matches 0.00227 & 0.00127 the columns labeled “infection rate” and “Attrib fraction in exposed (%) = 44.12” matches our “effectiveness” column.

# Credible? Incredible? A bayesian approach to how confident we are

The “problem” with a frequentist’s approach is that the “testing” framework

is rather contorted and you really can’t make the statements that you want to make.

We want to use the language of probability theory to say things like “there is an 90% chance that the vaccine is effective, which of course means there’s a 10% chance that it doesn’t”. As I have written before and many others have written elsewhere a *“p value”* and a *“confidence interval”* don’t give you that capability. Don’t get me wrong, the approach can be useful, is still the most frequent and dominant approach, but I don’t find it the best approach.

So let’s approach our warpspeed data using several different tools and allow ourselves the joy (okay I know that sounds geeky) to make probabilistic statements about what the data tell us. The rest of this post will be all about using

bayesian tools.

Since I’m always a fan of making use of existing packages and code that do what I want them to do I did some exploration looking to see what was available.

One of the first things I stumbled upon was a very nice function

bayes.prop.test

contained in a package

called bayesian\_first\_aid, sadly but not

surprisingly there was no CRAN version and development appears to have stopped circa 2015. Not surprising because there has been a lot more code

published for Bayesian methods since then and I know how much work it takes to keep up a package. There are others I’ll demonstrate a little later but the appeal of this function is it’s simplicity. It’s a great place to start

the code was easy to follow so I just grabbed the framework and updated it for my own needs. The respective R code are given below:

bayes\_binom\_test.R

|  |
| --- |
| bayes.binom.test <- function (x, n, comp.theta = 0.5, alternative = NULL, cred.mass = 0.95, n.iter=15000, progress.bar="none", p, conf.level) { |
|  |  |
|  | if(! missing(alternative)) { |
|  | warning("The argument 'alternative' is ignored by bayes.binom.test") |
|  | } |
|  |  |
|  | if(! missing(p)) { |
|  | comp.theta <- p |
|  | } |
|  |  |
|  | if(! missing(conf.level)) { |
|  | cred.mass <- conf.level |
|  | } |
|  |  |
|  | ### Begin code from binom.test |
|  | x\_name <- deparse(substitute(x)) |
|  | DNAME <- x\_name |
|  | n\_name <- deparse(substitute(n)) |
|  | xr <- round(x) |
|  | if (any(is.na(x) | (x < 0)) || max(abs(x - xr)) > 1e-07) |
|  | stop("'x' must be nonnegative and integer") |
|  | x <- xr |
|  | if (length(x) == 2L) { |
|  | n <- sum(x) |
|  | x <- x[1L] |
|  | } |
|  | else if (length(x) == 1L) { |
|  | nr <- round(n) |
|  | if ((length(n) > 1L) || is.na(n) || (n < 1) || abs(n - |
|  | nr) > 1e-07 || (x > nr)) |
|  | stop("'n' must be a positive integer >= 'x'") |
|  | DNAME <- paste(x\_name, "and", n\_name) |
|  | n <- nr |
|  | } |
|  | else stop("incorrect length of 'x'") |
|  | if (!missing(comp.theta) && (length(comp.theta) > 1L || is.na(comp.theta) || comp.theta < 0 || |
|  | comp.theta > 1)) |
|  | stop("'comp.theta' or 'p' must be a single number between 0 and 1") |
|  | if (!((length(cred.mass) == 1L) && is.finite(cred.mass) && |
|  | (cred.mass > 0) && (cred.mass < 1))) |
|  | stop("'cred.mass' or 'conf.level' must be a single number between 0 and 1") |
|  | ### END code from binom.test |
|  |  |
|  | mcmc\_samples <- jags\_binom\_test(x, n, n.chains = 3, n.iter = ceiling(n.iter / 3) , progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = comp.theta) |
|  | bfa\_object <- list(x = x, n = n, comp\_theta = comp.theta, cred\_mass = cred.mass, |
|  | x\_name = x\_name, n\_name = n\_name, data\_name = DNAME, |
|  | mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_binom\_test", "bayesian\_first\_aid") |
|  | bfa\_object |
|  | } |
|  |  |
|  |  |
|  | binom\_model\_string <- "model { |
|  | x ~ dbinom(theta, n) |
|  | theta ~ dbeta(1, 1) |
|  | x\_pred ~ dbinom(theta, n) |
|  | }" |
|  |  |
|  | jags\_binom\_test <- function(x, n, n.chains=3, n.iter=5000, progress.bar="none") { |
|  | mcmc\_samples <- run\_jags(binom\_model\_string, data = list(x = x, n = n), inits = list(theta = (x + 1) / (n + 2)), |
|  | params = c("theta", "x\_pred"), n.chains = n.chains, n.adapt = 0, |
|  | n.update = 0, n.iter = n.iter, thin = 1, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | ### binom test S3 methods ### |
|  |  |
|  | #' @export |
|  | print.bayes\_binom\_test <- function(x, ...) { |
|  |  |
|  | s <- format\_stats(x$stats)["theta",] |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian First Aid binomial test\n") |
|  | cat("\n") |
|  | cat("data: ", x$data\_name, "\n", sep="") |
|  | cat("number of successes = ", x$x,", number of trials = ", x$n, "\n", sep="") |
|  | cat("Estimated relative frequency of success:\n") |
|  | cat(" ", s["median"], "\n") |
|  | cat(s["HDI%"],"% credible interval:\n", sep="") |
|  | cat(" ", s[ c("HDIlo", "HDIup")], "\n") |
|  | cat("The relative frequency of success is more than", s["comp"] , "by a probability of", s["%>comp"], "\n") |
|  | cat("and less than", s["comp"] , "by a probability of", s["%<comp"], "\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | summary.bayes\_binom\_test <- function(object, ...) { |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | cat("number of successes = ", object$x,", number of trials = ", object$n, "\n", sep="") |
|  | cat("\n") |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("theta: the relative frequency of success\n") |
|  | cat("x\_pred: predicted number of successes in a replication\n") |
|  | cat("\n") |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_binom\_test |
|  | #' @export |
|  | plot.bayes\_binom\_test <- function(x, ...) { |
|  | old\_par <- par( mar=c(3.5,3.5,2.5,0.5) , mgp=c(2.25,0.7,0), mfcol=c(2,1)) |
|  | sample\_mat <- as.matrix(x$mcmc\_samples) |
|  | plotPost(sample\_mat[, "theta"], cred\_mass= x$cred\_mass, comp\_val=x$comp\_theta, xlim=c(0, 1), cex=1, cex.lab=1.5, |
|  | main = "Relative Frequency of Success", xlab=expression(theta), show\_median= TRUE) |
|  | hist\_data <- discrete\_hist(sample\_mat[, "x\_pred"], c(0, x$n), ylab="Probability", x\_marked= x$x, |
|  | xlab = "Number of sucesses",main="Data w. Post. Pred.") |
|  | #legend("topright", legend="Data", col="red", lty=1, lwd=3) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_binom\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("theta: The relative frequency of success\n") |
|  | cat("x\_pred: Predicted number of successes in a replication\n") |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.6) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | model.code.bayes\_binom\_test <- function(fit) { |
|  | cat("### Model code for the Bayesian First Aid alternative to the binomial test ###\n") |
|  | cat("require(rjags)\n\n") |
|  |  |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x, "\n") |
|  | cat("n <-", fit$n, "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(binom\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | binom\_model\_code <- function(x, n) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = list(x = x, n = n), |
|  | n.chains = 3, n.adapt=1000) |
|  | samples <- coda.samples(model, c("theta", "x\_pred"), n.iter=5000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | binom\_model\_code <- inject\_model\_string(binom\_model\_code, binom\_model\_string) |

Bayes\_cor\_test.R

|  |
| --- |
| bayes.cor.test <- function(x, ...) { |
|  | UseMethod("bayes.cor.test") |
|  | } |
|  |  |
|  | cor\_model\_string <- "model { |
|  | for(i in 1:n) { |
|  | xy[i,1:2] ~ dmt(mu[], prec[ , ], nu) |
|  | } |
|  |  |
|  | xy\_pred[1:2] ~ dmt(mu[], prec[ , ], nu) |
|  |  |
|  | # JAGS parameterizes the multivariate t using precision (inverse of variance) |
|  | # rather than variance, therefore here inverting the covariance matrix. |
|  | prec[1:2,1:2] <- inverse(cov[,]) |
|  |  |
|  | # Constructing the covariance matrix |
|  | cov[1,1] <- sigma[1] \* sigma[1] |
|  | cov[1,2] <- sigma[1] \* sigma[2] \* rho |
|  | cov[2,1] <- sigma[1] \* sigma[2] \* rho |
|  | cov[2,2] <- sigma[2] \* sigma[2] |
|  |  |
|  | # Priors |
|  | rho ~ dunif(-1, 1) |
|  | sigma[1] ~ dunif(sigmaLow, sigmaHigh) |
|  | sigma[2] ~ dunif(sigmaLow, sigmaHigh) |
|  | mu[1] ~ dnorm(mean\_mu, precision\_mu) |
|  | mu[2] ~ dnorm(mean\_mu, precision\_mu) |
|  | nu <- nuMinusOne+1 |
|  | nuMinusOne ~ dexp(1/29) |
|  | }" |
|  |  |
|  | jags\_cor\_test <- function(x, y, n.adapt= 500, n.chains=3, n.update = 100, n.iter=5000, thin=1, progress.bar="text") { |
|  |  |
|  | data\_list = list( |
|  | xy = cbind(x, y), |
|  | n = length(x), |
|  | mean\_mu = mean(c(x, y), trim=0.2) , |
|  | precision\_mu = 1 / (max(mad0(x), mad0(y))^2 \* 1000000), |
|  | sigmaLow = min(mad0(x), mad0(y)) / 1000 , |
|  | sigmaHigh = max(mad0(x), mad0(y)) \* 1000) |
|  |  |
|  | # Use robust estimates of the parameters as initial values |
|  | inits\_list = list(mu=c(mean(x, trim=0.2), mean(y, trim=0.2)), rho=cor(x, y, method="spearman"), |
|  | sigma = c(mad0(x), mad0(y)), nuMinusOne = 5) |
|  | mcmc\_samples <- run\_jags(cor\_model\_string, data = data\_list, inits = inits\_list, |
|  | params = c("rho", "mu", "sigma", "nu", "xy\_pred"), n.chains = n.chains, n.adapt = n.adapt, |
|  | n.update = n.update, n.iter = n.iter, thin = thin, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | #' @method bayes.cor.test default |
|  | #' @export |
|  | #' @rdname bayes.cor.test |
|  | bayes.cor.test.default <- function (x, y, alternative = c("two.sided", "less", "greater"), |
|  | method = c("pearson", "kendall", "spearman"), exact = NULL, |
|  | cred.mass = 0.95, continuity = FALSE, n.iter = 15000, progress.bar="text", |
|  | conf.level, ...) |
|  | { |
|  |  |
|  | if(! missing(conf.level)) { |
|  | cred.mass <- conf.level |
|  | } |
|  |  |
|  | if(! missing(alternative)) { |
|  | warning("The argument 'alternative' is ignored by bayes.binom.test") |
|  | } |
|  |  |
|  | if(! missing(exact)) { |
|  | warning("The argument 'exact' is ignored by bayes.binom.test") |
|  | } |
|  |  |
|  | if(! missing(continuity)) { |
|  | warning("The argument 'continuity' is ignored by bayes.binom.test") |
|  | } |
|  |  |
|  | ### BEGIN code from cor.test.default ### |
|  | alternative <- match.arg(alternative) |
|  | method <- match.arg(method) |
|  | x\_name <- deparse(substitute(x)) |
|  | y\_name <- deparse(substitute(y)) |
|  | data\_name <- paste(x\_name, "and", y\_name) |
|  | if (length(x) != length(y)) |
|  | stop("'x' and 'y' must have the same length") |
|  | if (!is.numeric(x)) |
|  | stop("'x' must be a numeric vector") |
|  | if (!is.numeric(y)) |
|  | stop("'y' must be a numeric vector") |
|  | # removes uncomplete pairs, this shouldn't be neccessary if JAGS could handle missing data in dmvt |
|  | OK <- complete.cases(x, y) |
|  | x <- x[OK] |
|  | y <- y[OK] |
|  | n <- length(x) |
|  | if (n < 3L) |
|  | stop("not enough observations. Need at least three complete observation.") |
|  | ### END code from cor.test.default |
|  | if (method == "kendall" || method == "spearman") { |
|  | stop("no non-parametric correlation comparable to Kendall's tau or Spearman's rho has been implemented yet.") |
|  | } |
|  | mcmc\_samples <- jags\_cor\_test(x, y, n.chains=3, n.iter=ceiling(n.iter / 3), thin=1, progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = 0) |
|  | bfa\_result <- list(x = x, y = y, cred\_mass = cred.mass, x\_name = x\_name, y\_name = y\_name, |
|  | data\_name = data\_name, x\_data\_expr = x\_name, y\_data\_expr = y\_name, |
|  | mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_result) <- c("bayes\_cor\_test", "bayesian\_first\_aid") |
|  | bfa\_result |
|  |  |
|  | } |
|  |  |
|  | #' @method bayes.cor.test formula |
|  | #' @export |
|  | #' @rdname bayes.cor.test |
|  | bayes.cor.test.formula <- function (formula, data, subset, na.action, ...) |
|  | { |
|  | ### BEGIN code from cor.test.formula ### |
|  | if (missing(formula) || !inherits(formula, "formula") || |
|  | length(formula) != 2L) |
|  | stop("'formula' missing or invalid") |
|  | m <- match.call(expand.dots = FALSE) |
|  | if (is.matrix(eval(m$data, parent.frame()))) |
|  | m$data <- as.data.frame(data) |
|  | m[[1L]] <- as.name("model.frame") |
|  | m$... <- NULL |
|  | mf <- eval(m, environment(formula)) |
|  | if (length(mf) != 2L) |
|  | stop("invalid formula") |
|  | DNAME <- paste(names(mf), collapse = " and ") |
|  | x\_name <- names(mf)[1] |
|  | y\_name <- names(mf)[2] |
|  | names(mf) <- c("x", "y") |
|  | ### END code from cor.test.formula ### |
|  |  |
|  | bfa\_result <- do.call("bayes.cor.test.default", c(mf, list(...))) |
|  | bfa\_result$data\_name <- DNAME |
|  | bfa\_result$x\_name <- x\_name |
|  | bfa\_result$y\_name <- y\_name |
|  | if(!missing(data)) { |
|  | data\_expr <- deparse(substitute(data)) |
|  | bfa\_result$x\_data\_expr <- paste(data\_expr, '[ , "', x\_name,'"]',sep="") |
|  | bfa\_result$y\_data\_expr <- paste(data\_expr, '[ , "', y\_name,'"]',sep="") |
|  | } else { |
|  | bfa\_result$x\_data\_expr <- x\_name |
|  | bfa\_result$y\_data\_expr <- y\_name |
|  | } |
|  |  |
|  | bfa\_result |
|  | } |
|  |  |
|  | ### Cor test S3 methods ### |
|  |  |
|  | #' @export |
|  | print.bayes\_cor\_test <- function(x, ...) { |
|  | s <- format\_stats(x$stats)["rho",] |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian First Aid Pearson's Correlation Coefficient Test\n") |
|  | cat("\n") |
|  | cat("data: ", x$data\_name, " (n = ", length(x$x) ,")\n", sep="") |
|  | cat("Estimated correlation:\n") |
|  | cat(" ", s["median"], "\n") |
|  | cat(s["HDI%"],"% credible interval:\n", sep="") |
|  | cat(" ", s[ c("HDIlo", "HDIup")], "\n") |
|  | cat("The correlation is more than", s["comp"] , "by a probability of", s["%>comp"], "\n") |
|  | cat("and less than", s["comp"] , "by a probability of", s["%<comp"], "\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | print\_bayes\_cor\_test\_params <- function(object) { |
|  | cat(" Model parameters\n") |
|  | cat("rho: the correlation between", object$data\_name, "\n") |
|  | cat("mu[1]: the mean of", object$x\_name, "\n") |
|  | cat("sigma[1]: the scale of", object$x\_name,", a consistent\n estimate of SD when nu is large.\n") |
|  | cat("mu[2]: the mean of", object$y\_name, "\n") |
|  | cat("sigma[2]: the scale of", object$y\_name,"\n") |
|  | cat("nu: the degrees-of-freedom for the bivariate t distribution\n") |
|  | cat("xy\_pred[1]: the posterior predictive distribution of", object$x\_name, "\n") |
|  | cat("xy\_pred[2]: the posterior predictive distribution of", object$y\_name, "\n") |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_cor\_test <- function(object, ...) { |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | cat(object$data\_name, ", n = ", length(object$x) ,"\n", sep="") |
|  | cat("\n") |
|  |  |
|  | print\_bayes\_cor\_test\_params(object) |
|  |  |
|  | cat("\n") |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' Summary plot of a bayes.cor.test object |
|  | #' |
|  | #' Produces a histogram of the posterior of the correlation parameter rho, and a scatterplot with the data and the posterior predictive density. |
|  | #' |
|  | #' @param x the resulting object from a \code{\link{bayes.cor.test}} run |
|  | #' @param xlim,ylim the limits of the scatterplot, if NULL then reasonable limits will be calculated automatically. |
|  | #' |
|  | #' @method plot bayes\_cor\_test |
|  | #' @export |
|  | plot.bayes\_cor\_test <- function(x, xlim = NULL, ylim = NULL,...) { |
|  | stats <- x$stats |
|  | mcmc\_samples <- x$mcmc\_samples |
|  | samples\_mat <- as.matrix(mcmc\_samples) |
|  |  |
|  | rho <- samples\_mat[, "rho"] |
|  |  |
|  | old\_par <- par(no.readonly = TRUE) |
|  |  |
|  | zones <- matrix(c(1,1,1, |
|  | 0,6,0, |
|  | 3,2,5, |
|  | 0,4,0), ncol = 3, byrow = TRUE) |
|  | layout(zones, widths=c(0.5,4,1), heights = c(6,3,10,1.5)) |
|  |  |
|  | ### fig 1, the posterior of rho ### |
|  | par(mar = c(2,2,2,2)) |
|  | plotPost(rho, comp\_val=0, cred\_mass=0.95, xlim=c(-1, 1), xlab="", main=expression(Correlation ~ (rho)), |
|  | show\_median=TRUE) |
|  |  |
|  |  |
|  | ### fig 2, the scatterplot ### |
|  |  |
|  | # Sampling from the posterior predictive distribution |
|  | # picking out 1000 samples and for each sample generate 100 samples from the corresponding |
|  | # bivariate t distribution. (This is faster than picking out 100000 samples and generating) |
|  | # one bivariate t sample per sample. |
|  | xy\_rep <- do.call(rbind, lapply(sample(1:nrow(samples\_mat), 1000, replace=T), function(i) { |
|  | sigma1 <- samples\_mat[i, "sigma[1]"] |
|  | sigma2 <- samples\_mat[i, "sigma[2]"] |
|  | rho <- samples\_mat[i, "rho"] |
|  | cov\_mat <- cbind(c(sigma1^2, sigma1 \* sigma2 \* rho), |
|  | c(sigma1 \* sigma2 \* rho, sigma2^2)) |
|  | rmt(100, samples\_mat[i, c("mu[1]", "mu[2]")], cov\_mat, samples\_mat[i, "nu"]) |
|  | })) |
|  |  |
|  |  |
|  | x\_rep <- xy\_rep[,1] |
|  | y\_rep <- xy\_rep[,2] |
|  | # Calculating the 2d density of the posterior predictive distribution x\_rep and y\_rep |
|  | dens\_limits <- c(median(x\_rep) - IQR(x\_rep) \* 5, median(x\_rep) + IQR(x\_rep) \* 5, |
|  | median(y\_rep) - IQR(y\_rep) \* 5, median(y\_rep) + IQR(y\_rep) \* 5) |
|  | bandwidth <- c(IQR(x\_rep), IQR(y\_rep)) / 1.349 \* 0.5 |
|  | dens\_2d <- kde2d(x\_rep, y\_rep, n = 40, lims=dens\_limits, h=bandwidth) |
|  | sorted\_z <- sort(dens\_2d$z, decreasing=TRUE) |
|  | post\_95\_limit <- sorted\_z[which.min(abs(cumsum(sorted\_z / sum(sorted\_z)) - 0.95))] |
|  | post\_50\_limit <- sorted\_z[which.min(abs(cumsum(sorted\_z / sum(sorted\_z)) - 0.5))] |
|  | # These messy lines calculates limits of the plot that makes sure both all the data |
|  | # and the density estimate is visible. Also centers the plot on the median of the data. |
|  | if(is.null(xlim)) { |
|  | plot\_xlim <- c(median(x$x) - max( abs(x$x - median(x$x))), median(x$x) + max( abs(x$x - median(x$x)))) |
|  | plot\_xlim[1] <- min(plot\_xlim[1], dens\_2d$x[apply(dens\_2d$z > post\_95\_limit, 2, any)] - diff(dens\_2d$x[1:2]) / 2) |
|  | plot\_xlim[2] <- max(plot\_xlim[2], dens\_2d$x[apply(dens\_2d$z > post\_95\_limit, 2, any)] + diff(dens\_2d$x[1:2])/ 2) |
|  | } else { |
|  | plot\_xlim <- xlim |
|  | } |
|  | if(is.null(ylim)) { |
|  | plot\_ylim <- c(median(x$y) - max( abs(x$y - median(x$y))), median(x$y) + max( abs(x$y - median(x$y)))) |
|  | plot\_ylim[1] <- min(plot\_ylim[1], dens\_2d$y[apply(dens\_2d$z > post\_95\_limit, 1, any)] - diff(dens\_2d$y[1:2])/ 2) |
|  | plot\_ylim[2] <- max(plot\_ylim[2], dens\_2d$y[apply(dens\_2d$z > post\_95\_limit, 1, any)] + diff(dens\_2d$y[1:2])/ 2) |
|  | } else { |
|  | plot\_ylim <- ylim |
|  | } |
|  |  |
|  | #### Code for plotting coverage ellipses, doesn't look as good though as the kernel density version. |
|  | #### Commented out until I find a better solution... |
|  | # xy\_rep <- samples\_mat[,c("xy\_pred[1]", "xy\_pred[2]")] |
|  | # # Use a subset of the posterior predictive to calculate higest density ellipses |
|  | # # At most 2000 points |
|  | # #xy\_rep <- xy\_rep[seq(1, nrow(xy\_rep), length.out = min(nrow(xy\_rep), 5000)),] |
|  | # |
|  | # # Find the points contained in the minimum ellipse covering 50%/95% of the points |
|  | # # Unfortunately cov.mve doesn't return the actuall ellipse (but another ellipse). |
|  | # # Then finding the minimum volume ellipse that contains these 50%/95% points |
|  | # points\_95\_perc <- xy\_rep[cov.rob(xy\_rep, quantile.used = round(nrow(xy\_rep) \* 0.95), method = "mve", nsamp = 6000)$best, ] |
|  | # ellipse\_95\_perc <- predict(ellipsoidhull( points\_95\_perc), n.out = 100)) |
|  | # |
|  | # # Using only the 95% points in the highest density ellipse thus the "0.5 / 0.95" |
|  | # points\_50\_perc <- points\_95\_perc[cov.rob(points\_95\_perc, quantile.used = round(nrow(points\_95\_perc) \* 0.5 / 0.95), method = "mve", nsamp = 6000)$best, ] |
|  | # ellipse\_50\_perc <- predict(ellipsoidhull( points\_50\_perc), n.out = 100)) |
|  | # # These messy lines calculates limits of the plot that makes sure both all the data |
|  | # # and the ellipses are visible. Also centers the plot on the median of the data. |
|  | # x\_dist\_from\_median <- max( abs(c(x$x, points\_95\_perc[,1]) - median(x$x)) ) |
|  | # plot\_xlim <- c(median(x$x) - x\_dist\_from\_median, median(x$x) + x\_dist\_from\_median) |
|  | # y\_dist\_from\_median <- max( abs(c(x$y, points\_95\_perc[,2]) - median(x$y)) ) |
|  | # plot\_ylim <- c(median(x$y) - y\_dist\_from\_median, median(x$y) + y\_dist\_from\_median) |
|  |  |
|  | # Plotting |
|  | par(mar = c(2,2,0,0), xaxt="s", yaxt="s", bty="o") |
|  | plot(x$x, x$y, col=rgb(1,1,1, 0), xlim=plot\_xlim, ylim=plot\_ylim) |
|  | .filled.contour(dens\_2d$x, dens\_2d$y, dens\_2d$z, levels=c(0, post\_95\_limit, post\_50\_limit, max(sorted\_z)), col=c(rgb(1,1,1, 0), "#bddeeb", "#87ceeb")) |
|  |  |
|  | # Plotting the ellipses |
|  | # polygon(ellipse\_95\_perc, col = "#bddeeb", border = NA) |
|  | # polygon(ellipse\_50\_perc, col = "#87ceeb", border = NA) |
|  | points(x$x, x$y) |
|  |  |
|  | # Save the limits for use with the marginal plots |
|  | scatter\_lim <- par("usr") |
|  |  |
|  | ### The titles of the scatter plot ### |
|  |  |
|  | par(xaxt="n", yaxt="n",bty="n", mar = c(.3,2,.3,0) +.05) |
|  | # fig 3 |
|  | plot(x=1,y=1,type="n",ylim=c(-1,1), xlim=c(-1,1)) |
|  | text(0,0, x$y\_name, cex=1.5, srt=90) |
|  | # fig 4 |
|  | plot(x=1,y=1,type="n",ylim=c(-1,1), xlim=c(-1,1)) |
|  | text(0,0, x$x\_name, cex=1.5) |
|  |  |
|  | ### fig 5, the y histogram ### |
|  | par(mar = c(2,0,0,1)) |
|  |  |
|  | i <- sample(1:nrow(samples\_mat), 20) |
|  | y\_mu <- samples\_mat[i, "mu[2]"] |
|  | y\_sigma <- samples\_mat[i, "sigma[2]"] |
|  | y\_nu <- samples\_mat[i, "nu"] |
|  |  |
|  | hist\_with\_t\_curves(x$y, stats["mu[2]", "mean"], stats["sigma[2]", "mean"], y\_mu, y\_sigma, y\_nu, |
|  | axes = FALSE, horiz=TRUE, plot\_n=FALSE, x\_lim=scatter\_lim[3:4], axs="i") |
|  |  |
|  | ### fig 6, The x histogram ### |
|  | par(mar = c(0,2,1,0)) |
|  |  |
|  | i <- sample(1:nrow(samples\_mat), 20) |
|  | x\_mu <- samples\_mat[i, "mu[1]"] |
|  | x\_sigma <- samples\_mat[i, "sigma[1]"] |
|  | x\_nu <- samples\_mat[i, "nu"] |
|  |  |
|  | hist\_with\_t\_curves(x$x,stats["mu[1]", "mean"], stats["sigma[1]", "mean"], x\_mu, x\_sigma, y\_nu, |
|  | horiz=FALSE, plot\_n=TRUE, axes = FALSE, x\_lim=scatter\_lim[1:2], axs="i") |
|  |  |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_cor\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | print\_bayes\_cor\_test\_params(fit) |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_cor\_test <- function(fit) { |
|  | cat("## Model code for the Bayesian First Aid alternative to Pearson's correlation test. ##\n") |
|  |  |
|  | cat("require(rjags)\n\n") |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x\_data\_expr, "\n") |
|  | cat("y <-", fit$y\_data\_expr, "\n") |
|  | cat("xy <- cbind(x, y)\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(cor\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | cor\_model\_code <- function(x, y, xy) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Initializing the data list and setting parameters for the priors |
|  | # that in practice will result in flat priors on mu and sigma. |
|  | data\_list = list( |
|  | xy = xy, |
|  | n = length(x), |
|  | mean\_mu = mean(c(x, y), trim=0.2) , |
|  | precision\_mu = 1 / (max(mad(x), mad(y))^2 \* 1000000), |
|  | sigmaLow = min(mad(x), mad(y)) / 1000 , |
|  | sigmaHigh = max(mad(x), mad(y)) \* 1000) |
|  |  |
|  | # Initializing parameters to sensible starting values helps the convergence |
|  | # of the MCMC sampling. Here using robust estimates of the mean (trimmed) |
|  | # and standard deviation (MAD). |
|  | inits\_list = list(mu=c(mean(x, trim=0.2), mean(y, trim=0.2)), rho=cor(x, y, method="spearman"), |
|  | sigma = c(mad(x), mad(y)), nuMinusOne = 5) |
|  |  |
|  | # The parameters to monitor. |
|  | params <- c("rho", "mu", "sigma", "nu", "xy\_pred") |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = data\_list, |
|  | inits = inits\_list, n.chains = 3, n.adapt=1000) |
|  | update(model, 500) # Burning some samples to the MCMC gods.... |
|  | samples <- coda.samples(model, params, n.iter=5000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | cor\_model\_code <- inject\_model\_string(cor\_model\_code, cor\_model\_string) |

Bayes\_poission\_test.R

|  |
| --- |
| bayes.poisson.test <- function (x, T = 1, r = 1, alternative = c("two.sided", "less", "greater"), |
|  | cred.mass = 0.95, n.iter = 15000, progress.bar="none", conf.level) |
|  | { |
|  |  |
|  | if(! missing(alternative)) { |
|  | warning("The argument 'alternative' is ignored by bayes.poisson.test") |
|  | } |
|  |  |
|  | if(! missing(conf.level)) { |
|  | cred.mass <- conf.level |
|  | } |
|  |  |
|  | x\_name <- deparse(substitute(x)) |
|  | t\_name <- deparse(substitute(T)) |
|  |  |
|  | ### BEGIN Slightly modified code from poisson.test ### |
|  | DNAME <- deparse(substitute(x)) |
|  | DNAME <- paste(DNAME, ", time base: ", deparse(substitute(T)), sep = "") |
|  | if ((l <- length(x)) != length(T)) |
|  | if (length(T) == 1L) |
|  | T <- rep(T, l) |
|  | else stop("'x' and 'T' have incompatible length") |
|  | xr <- round(x) |
|  | if (any(!is.finite(x) | (x < 0)) || max(abs(x - xr)) > 1e-07) |
|  | stop("'x' must be finite, nonnegative, and integer") |
|  | x <- xr |
|  | if (any(is.na(T) | (T < 0))) |
|  | stop("'T' must be nonnegative") |
|  | if ((k <- length(x)) < 1L) |
|  | stop("not enough data") |
|  | if (k > 2L) |
|  | stop(paste("The case for more than two groups is unimplemented.", |
|  | "Run the model with just two groups and use the model.code function", |
|  | "to print the code that implements the model. This code can then be", |
|  | "easily modified to work with many groups!", sep="\n")) |
|  | if (!missing(r) && (length(r) > 1 || is.na(r) || r < 0)) |
|  | stop("'r' must be a single positive number") |
|  | alternative <- match.arg(alternative) |
|  | ### END code from poisson.test ### |
|  |  |
|  | if (k == 2) { |
|  | # two samle poison test |
|  | mcmc\_samples <- jags\_two\_sample\_poisson\_test(x[1], T[1], x[2], T[2], |
|  | n.chains=3, n.iter= ceiling(n.iter / 3), progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = r) |
|  |  |
|  | # Calculating the rate\_ratio mean and HDI on the log scale. |
|  | ratio\_stats <- mcmc\_stats(log(as.matrix(mcmc\_samples)[,"rate\_ratio"]), cred\_mass = cred.mass, comp\_val = log(r)) |
|  | stats["rate\_ratio", c("mean", "HDIlo", "HDIup")] <- exp(ratio\_stats[,c("mean", "HDIlo", "HDIup")]) |
|  | stats["rate\_ratio", "sd"] <- NA # Setting sd to NA, as it is clear what should be presented |
|  | # when the mean was calculated on the log transformed rate\_ratio |
|  |  |
|  | bfa\_object <- list(x = x, t = T, r = r, cred\_mass = cred.mass, x\_name = x\_name, t\_name = t\_name, |
|  | data\_name = DNAME, mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_two\_sample\_poisson\_test", "bayesian\_first\_aid") |
|  | } |
|  | else { # k == 1 |
|  | # one samle poison test |
|  | mcmc\_samples <- jags\_one\_sample\_poisson\_test(x, T, n.chains=3, n.iter= ceiling(n.iter / 3), progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = r) |
|  | bfa\_object <- list(x = x, t = T, r = r, cred\_mass = cred.mass, x\_name = x\_name, t\_name = t\_name, |
|  | data\_name = DNAME, mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_one\_sample\_poisson\_test", "bayesian\_first\_aid") |
|  | } |
|  | bfa\_object |
|  | } |
|  |  |
|  | one\_sample\_poisson\_model\_string <- "model { |
|  | x ~ dpois(lambda \* t) |
|  | lambda ~ dgamma(0.5, 0.00001) |
|  | x\_pred ~ dpois(lambda \* t) |
|  | }" |
|  |  |
|  | jags\_one\_sample\_poisson\_test <- function(x, t, n.chains, n.iter, progress.bar) { |
|  | mcmc\_samples <- run\_jags(one\_sample\_poisson\_model\_string, data = list(x = x, t = t), inits = list(lambda = (x + 0.5) / t), |
|  | params = c("lambda", "x\_pred"), n.chains = n.chains, n.adapt = 0, |
|  | n.update = 100, n.iter = n.iter, thin = 1, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  |  |
|  | two\_sample\_poisson\_model\_string <- "model { |
|  | for(group\_i in 1:2) { |
|  | x[group\_i] ~ dpois(lambda[group\_i] \* t[group\_i]) |
|  | lambda[group\_i] ~ dgamma(0.5, 0.00001) |
|  | x\_pred[group\_i] ~ dpois(lambda[group\_i] \* t[group\_i]) |
|  | } |
|  | rate\_diff <- lambda[1] - lambda[2] |
|  | rate\_ratio <- lambda[1] / lambda[2] |
|  | }" |
|  |  |
|  | jags\_two\_sample\_poisson\_test <- function(x1, t1, x2, t2, n.chains, n.iter, progress.bar) { |
|  | data\_list = list(x = c(x1, x2), t = c(t1, t2)) |
|  | init\_list = list(lambda = ( c(x1, x2) + 0.5 ) / c(t1, t2) ) |
|  | mcmc\_samples <- run\_jags(two\_sample\_poisson\_model\_string, data = data\_list, inits = init\_list, |
|  | params = c("lambda", "x\_pred","rate\_diff", "rate\_ratio"), n.chains = n.chains, n.adapt = 0, |
|  | n.update = 100, n.iter = n.iter, thin = 1, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | # ... |
|  |  |
|  |  |
|  | ### One sample poisson test S3 methods ### |
|  |  |
|  | #' @export |
|  | print.bayes\_one\_sample\_poisson\_test <- function(x, ...) { |
|  | s <- format\_stats(x$stats) |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian Fist Aid poisson test - one sample\n") |
|  | cat("\n") |
|  | cat("number of events: ", x$x, ", time periods: ", x$t, sep="") |
|  | cat("\n") |
|  | cat("Estimated event rate:\n") |
|  | cat(" ", s["lambda", "median"], "\n") |
|  | cat(s["lambda","HDI%"],"% credible interval:\n", sep="") |
|  | cat(" ", s["lambda", c("HDIlo", "HDIup")], "\n") |
|  | cat("The event rate is more than", s["lambda", "comp"] , "by a probability of", s["lambda", "%>comp"], "\n") |
|  | cat("and less than", s["lambda", "comp"] , "by a probability of", s["lambda", "%<comp"], " .\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_one\_sample\_poisson\_test <- function(object, ...) { |
|  | s <- round\_or\_signif(object$stats, 3) |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("lambda: the rate of the process.\n") |
|  | cat("x\_pred: predicted event count during", object$t ,"periods.\n") |
|  | cat("\n") |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_one\_sample\_poisson\_test |
|  | #' @export |
|  | plot.bayes\_one\_sample\_poisson\_test <- function(x, ...) { |
|  | old\_par <- par( mar=c(3.5,3.5,2.5,0.5) , mgp=c(2.25,0.7,0), mfcol=c(2,1)) |
|  | sample\_mat <- as.matrix(x$mcmc\_samples) |
|  | lambda <- sample\_mat[, "lambda"] |
|  | # Calculating the xlim to include the comparison rate and 0 \*unless\* they are too |
|  | # far away from the samples. |
|  | xlim = range(lambda) |
|  | if(0 > xlim[1] - diff(range(lambda)) / 2) { |
|  | xlim[1] <- 0 |
|  | } |
|  | if(x$r > xlim[1] - diff(range(lambda)) / 2) { |
|  | xlim[1] <- min(x$r, xlim[1]) |
|  | } |
|  | if(x$r < xlim[2] + diff(range(lambda)) / 2) { |
|  | xlim[2] <- max(x$r, xlim[2]) |
|  | } |
|  |  |
|  | plotPost(lambda, cred\_mass= x$cred\_mass, comp\_val=x$r, cex=1, cex.lab=1.5, |
|  | xlim=xlim, main = "Rate of occurence", xlab=expression(lambda), show\_median= TRUE) |
|  | hist\_data <- discrete\_hist(sample\_mat[, "x\_pred"], c(0, max(sample\_mat[, "x\_pred"])), ylab="Probability", x\_marked= x$x, |
|  | xlab = paste("Event count during", x$t, "periods"), main="Data w. Post. Pred.") |
|  | #legend("topright", legend="Data", col="red", lty=1, lwd=3) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_one\_sample\_poisson\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("lambda: the rate of the process.\n") |
|  | cat("x\_pred: predicted event count during", fit$t ,"periods.\n") |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_one\_sample\_poisson\_test <- function(fit) { |
|  | cat("### Model code for the Bayesian First Aid one sample Poisson test ###\n") |
|  | cat("require(rjags)\n\n") |
|  |  |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x, "\n") |
|  | cat("t <-", fit$t, "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(one\_sample\_poisson\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | one\_sample\_poisson\_model\_code <- function(x, t) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = list(x = x, t = t), n.chains = 3) |
|  | samples <- coda.samples(model, c("lambda", "x\_pred"), n.iter=5000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | one\_sample\_poisson\_model\_code <- inject\_model\_string(one\_sample\_poisson\_model\_code, one\_sample\_poisson\_model\_string) |
|  |  |
|  | ### Two sample poisson test S3 methods ### |
|  |  |
|  |  |
|  | #' @export |
|  | print.bayes\_two\_sample\_poisson\_test <- function(x, ...) { |
|  | s <- format\_stats(x$stats) |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian Fist Aid poisson test - two sample\n") |
|  | cat("\n") |
|  | cat("number of events: ", paste(x$x[1], " and ", x$x[2], sep=""), ", time periods: ", paste(x$t[1], " and ", x$t[2], sep=""), "\n", sep="") |
|  | cat("\n") |
|  | cat(" Estimates [", s[1, "HDI%"] ,"% credible interval]\n", sep="") |
|  | cat("Group 1 rate: ", s["lambda[1]", "median"], " [", s["lambda[1]", "HDIlo"], ", ", s["lambda[1]", "HDIup"], "]","\n", sep="") |
|  | cat("Group 2 rate: ", s["lambda[2]", "median"], " [", s["lambda[2]", "HDIlo"], ", ", s["lambda[2]", "HDIup"], "]","\n", sep="") |
|  | cat("Rate ratio (Group 1 rate / Group 2 rate):\n ", s["rate\_ratio", "median"], " [", s["rate\_ratio", "HDIlo"], ", ", s["rate\_ratio", "HDIup"], "]","\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat("The event rate of group 1 is more than", s["rate\_ratio", "comp"], "times that of group 2 by a probability", "\n") |
|  | cat("of", s["rate\_ratio", "%>comp"], "and less than", s["rate\_ratio", "comp"], "times that of group 2 by a probability of", s["rate\_ratio", "%<comp"], ".\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | print\_bayes\_two\_sample\_poisson\_test\_params <- function(x) { |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("lambda[1]: the rate of the process of group 1.\n") |
|  | cat("lambda[2]: the rate of the process of group 2.\n") |
|  | cat("x\_pred[1]: predicted event count of group 1 during", x$t[1] ,"periods.\n") |
|  | cat("x\_pred[2]: predicted event count of group 2 during", x$t[2] ,"periods.\n") |
|  | cat("rate\_diff: The difference lambda[1] - lambda[2].\n") |
|  | cat("rate\_ratio: The ratio lambda[1] / lambda[2].\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_two\_sample\_poisson\_test <- function(object, ...) { |
|  | s <- round\_or\_signif(object$stats, 3) |
|  |  |
|  | print\_bayes\_two\_sample\_poisson\_test\_params(object) |
|  | cat("\n") |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  | cat("For rate\_ratio the mean, 'HDIlo' and 'HDIup' are calculated on the log transformed\n", sep="") |
|  | cat("samples and then transformed back to the original scale.\n", sep="") |
|  |  |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_two\_sample\_poisson\_test |
|  | #' @export |
|  | plot.bayes\_two\_sample\_poisson\_test <- function(x, ...) { |
|  | old\_par <- par( mar=c(3.5,3.5,2.5,0.5) , mgp=c(2.25,0.7,0), mfcol=c(3,1)) |
|  | sample\_mat <- as.matrix(x$mcmc\_samples) |
|  | lambda1 <- sample\_mat[, "lambda[1]"] |
|  | lambda2 <- sample\_mat[, "lambda[2]"] |
|  |  |
|  | xlim <- range(lambda1, lambda2) |
|  | if(0 > xlim[1] - diff(range(lambda1, lambda2)) / 2) { |
|  | xlim[1] <- 0 |
|  | } |
|  | plotPost(lambda1, cred\_mass= x$cred\_mass, cex=1.5, cex.lab=1.8, xlim=xlim, |
|  | main = "Rate of occurence for group 1", xlab=expression(lambda[1]), show\_median= TRUE) |
|  | plotPost(lambda2, cred\_mass= x$cred\_mass, cex=1.5, cex.lab=1.8, xlim=xlim, |
|  | main = "Rate of occurence for group 2", xlab=expression(lambda[2]), show\_median= TRUE) |
|  |  |
|  | # Centers the ratio plot on 1.0. |
|  | xlim <- 2^(max(abs(range(log2(sample\_mat[, "rate\_ratio"])))) \* c(-1, 1)) |
|  | plotPost(sample\_mat[, "rate\_ratio"], cred\_mass= x$cred\_mass, comp\_val = x$r, cex=1.5, cex.lab=1.8, xlim=xlim, |
|  | main = "Rate ratio between group 1 and group 2", xlab=expression(lambda[1] / lambda[2]), show\_median= TRUE, log\_base = 2) |
|  |  |
|  |  |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_two\_sample\_poisson\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | print\_bayes\_two\_sample\_poisson\_test\_params(fit) |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_two\_sample\_poisson\_test <- function(fit) { |
|  | cat("### Model code for the Bayesian First Aid two sample Poisson test ###\n") |
|  | cat("require(rjags)\n\n") |
|  |  |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", deparse(fit$x), "\n") |
|  | cat("t <-", deparse(fit$t), "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(two\_sample\_poisson\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | two\_sample\_poisson\_model\_code <- function(x, t) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = list(x = x, t = t), n.chains = 3) |
|  | samples <- coda.samples(model, c("lambda", "x\_pred", "rate\_diff", "rate\_ratio"), n.iter=5000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | two\_sample\_poisson\_model\_code <- inject\_model\_string(two\_sample\_poisson\_model\_code, two\_sample\_poisson\_model\_string) |

Bayes\_prop\_test.R

|  |
| --- |
| bayes.prop.test <- function (x, n, comp.theta = NULL, alternative = NULL, cred.mass = 0.95, correct = NULL, n.iter=15000, progress.bar="none", p, conf.level) { |
|  |  |
|  | if(! missing(alternative)) { |
|  | warning("The argument 'alternative' is ignored by bayes.prop.test") |
|  | } |
|  |  |
|  | if(! missing(correct)) { |
|  | warning("The argument 'correct' is ignored by bayes.prop.test") |
|  | } |
|  |  |
|  | if(! missing(p)) { |
|  | comp.theta <- p |
|  | } |
|  |  |
|  | if(! missing(conf.level)) { |
|  | cred.mass <- conf.level |
|  | } |
|  |  |
|  | x\_name <- deparse(substitute(x)) |
|  | n\_name <- deparse(substitute(n)) |
|  |  |
|  | ### Begin slightly modified code from prop.test |
|  | DNAME <- deparse(substitute(x)) |
|  | if (is.table(x) && length(dim(x)) == 1L) { |
|  | if (dim(x) != 2L) |
|  | stop("table 'x' should have 2 entries") |
|  | l <- 1 |
|  | n <- sum(x) |
|  | x <- x[1L] |
|  | } |
|  | else if (is.matrix(x)) { |
|  | if (ncol(x) != 2L) |
|  | stop("'x' must have 2 columns") |
|  | l <- nrow(x) |
|  | n <- rowSums(x) |
|  | x <- x[, 1L] |
|  | } |
|  | else { |
|  | DNAME <- paste(DNAME, "out of", deparse(substitute(n))) |
|  | if ((l <- length(x)) != length(n)) |
|  | stop("'x' and 'n' must have the same length") |
|  | } |
|  | OK <- complete.cases(x, n) |
|  | x <- x[OK] |
|  | n <- n[OK] |
|  | if ((k <- length(x)) < 1L) |
|  | stop("not enough data") |
|  | if (any(n <= 0)) |
|  | stop("elements of 'n' must be positive") |
|  | if (any(x < 0)) |
|  | stop("elements of 'x' must be nonnegative") |
|  | if (any(x > n)) |
|  | stop("elements of 'x' must not be greater than those of 'n'") |
|  | if(length(comp.theta) == 1) { |
|  | comp.theta <- rep(comp.theta, length(x)) |
|  | } |
|  | if (is.null(comp.theta) && (k == 1)) |
|  | comp.theta <- 0.5 |
|  | if (!is.null(comp.theta)) { |
|  | if (length(comp.theta) != l) |
|  | stop("'comp.theta' must have the same length as 'x' and 'n' or be a single number") |
|  | comp.theta <- comp.theta[OK] |
|  | if (any((comp.theta <= 0) | (comp.theta >= 1))) |
|  | stop("elements of 'comp.theta' must be in (0,1)") |
|  | } |
|  | if ((length(cred.mass) != 1L) || is.na(cred.mass) || |
|  | (cred.mass <= 0) || (cred.mass >= 1)) |
|  | stop("'cred.mass' must be a single number between 0 and 1") |
|  |  |
|  | ### END code from prop.test |
|  |  |
|  | if(length(x) == 1) { |
|  | return(bayes.binom.test(x, n, comp.theta, cred.mass = ifelse(is.null(cred.mass), 0.5, cred.mass), |
|  | n.iter = n.iter, progress.bar = progress.bar)) |
|  | } |
|  |  |
|  | mcmc\_samples <- jags\_prop\_test(x, n, n.chains = 3, n.iter = ceiling(n.iter / 3) , progress.bar=progress.bar) |
|  |  |
|  | if(is.null(comp.theta)) { |
|  | temp\_comp\_val <- 0.5 |
|  | } else { |
|  | temp\_comp\_val <- comp.theta |
|  | } |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = temp\_comp\_val) |
|  | diff\_stats <- mcmc\_stats(create\_theta\_diff\_matrix(as.matrix(mcmc\_samples)), cred\_mass = cred.mass, comp\_val = 0) |
|  | stats <- rbind(stats, diff\_stats) |
|  | bfa\_object <- list(x = x, n = n, comp\_theta = comp.theta, cred\_mass = cred.mass, |
|  | x\_name = x\_name, n\_name = n\_name, data\_name = DNAME, |
|  | mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_prop\_test", "bayesian\_first\_aid") |
|  | bfa\_object |
|  | } |
|  |  |
|  | create\_theta\_diff\_matrix <- function(samples\_mat) { |
|  | n\_groups <- sum(str\_count(colnames(samples\_mat), "theta\\[")) |
|  | combs <- combn(n\_groups, 2) |
|  | theta\_diffs <- sapply(1:ncol(combs), function(comb\_i) { |
|  | i <- combs[1, comb\_i] |
|  | j <- combs[2, comb\_i] |
|  | theta\_diff <- samples\_mat[,paste0("theta[", i,"]")] - samples\_mat[,paste0("theta[", j,"]")] |
|  | theta\_diff <- matrix(theta\_diff, nrow = 1, dimnames = NULL) |
|  | theta\_diff |
|  | }) |
|  |  |
|  | colnames(theta\_diffs) <- apply(combs, 2, function(comb) {paste0("theta\_diff[", comb[1], ",", comb[2], "]")}) |
|  | theta\_diffs |
|  | } |
|  |  |
|  | prop\_model\_string <- "model { |
|  | for(i in 1:length(x)) { |
|  | x[i] ~ dbinom(theta[i], n[i]) |
|  | theta[i] ~ dbeta(1, 1) |
|  | x\_pred[i] ~ dbinom(theta[i], n[i]) |
|  | } |
|  | }" |
|  |  |
|  | jags\_prop\_test <- function(x, n, n.chains=3, n.iter=5000, progress.bar="none") { |
|  | mcmc\_samples <- run\_jags(prop\_model\_string, data = list(x = x, n = n), inits = list(theta = (x + 1) / (n + 2)), |
|  | params = c("theta", "x\_pred"), n.chains = n.chains, n.adapt = 0, |
|  | n.update = 0, n.iter = n.iter, thin = 1, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  |  |
|  | format\_group\_diffs <- function(bfa\_object) { |
|  | s <- bfa\_object$stats |
|  | n\_groups <- length(bfa\_object$x) |
|  | med\_diff\_mat <- matrix("", nrow = n\_groups, ncol = n\_groups) |
|  | hdi\_diff\_mat <- matrix("", nrow = n\_groups, ncol = n\_groups) |
|  | diff\_names <- rownames(s)[ str\_detect(rownames(s), "theta\_diff\\[")] |
|  | for(diff\_name in diff\_names) { |
|  | indices\_match <- str\_match(diff\_name, "\\[(\\d+),(\\d+)\\]$") |
|  | i <- as.numeric(indices\_match[1,2]) |
|  | j <- as.numeric(indices\_match[1,3]) |
|  | med\_diff\_mat[i, j] <- as.character(round(s[diff\_name, "median"], 2) ) |
|  | hdi\_diff\_mat[i, j] <- paste("[", signif(s[diff\_name, "HDIlo"], 2), ", ", signif(s[diff\_name, "HDIup"], 2), "]", sep="") |
|  | } |
|  | diff\_mat <- matrix("", nrow = n\_groups \* 2, ncol = n\_groups) |
|  | for(i in seq\_len(n\_groups)) { |
|  | diff\_mat[1 + (i - 1) \* 2,] <- med\_diff\_mat[i,] |
|  | diff\_mat[2 + (i - 1) \* 2,] <- hdi\_diff\_mat[i,] |
|  | } |
|  | rownames(diff\_mat) <- rep(seq\_len(n\_groups), each = 2) |
|  | rownames(diff\_mat)[1:nrow(diff\_mat) %% 2 == 0] <- "" |
|  | rownames(diff\_mat) <- paste0(" ", rownames(diff\_mat)) |
|  | diff\_mat <- format(diff\_mat, width = max(nchar(diff\_mat)), justify = "centre") |
|  | colnames(diff\_mat) <- format(as.character(1:ncol(diff\_mat)), width = max(nchar(diff\_mat)), justify = "centre") |
|  | diff\_mat <- diff\_mat[-c(nrow(diff\_mat) - 1, nrow(diff\_mat)), -1, drop=FALSE] |
|  | diff\_mat |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #' @method plot bayes\_prop\_test |
|  | #' @export |
|  | plot.bayes\_prop\_test <- function(x, ...) { |
|  | samples <- as.matrix(x$mcmc\_samples) |
|  | # Throw away everything except the what we want to plot, the theta samples. |
|  | samples <- samples[,str\_detect(colnames(samples), "^theta\\[")] |
|  | n\_groups <- length(x$x) |
|  | diff\_samples <- create\_theta\_diff\_matrix(as.matrix(x$mcmc\_samples)) |
|  | layout\_mat <- matrix( 0 , nrow=n\_groups, ncol=n\_groups) |
|  | #layout\_mat[,1] <- seq\_len(n\_groups) |
|  | diag(layout\_mat) <- seq\_len(n\_groups) |
|  |  |
|  | old\_par <- par(no.readonly = TRUE) |
|  | layout\_mat <- t(layout\_mat) |
|  | layout\_mat[lower.tri(layout\_mat)] <- seq(n\_groups + 1, by = 2,length.out = (ncol(diff\_samples))) |
|  | layout\_mat <- t(layout\_mat) |
|  | layout\_mat[lower.tri(layout\_mat)] <- seq(n\_groups + 2, by = 2,length.out = (ncol(diff\_samples))) |
|  | layout(layout\_mat) |
|  | par( mar=c(3.5,2,2,2) , mgp=c(2.25,0.7,0) ) |
|  | post\_xlim <- range(apply(samples, 2, quantile, probs = c(0.001, 0.999))) |
|  | # Some rules for making the post\_xlim nice, with a preference for showing endpoints of the scale |
|  | xlim\_length <- abs(diff(post\_xlim)) |
|  | if( post\_xlim[1] - xlim\_length < 0) { |
|  | post\_xlim[1] <- 0 |
|  | } |
|  | if(post\_xlim[2] + xlim\_length > 1) { |
|  | post\_xlim[2] <- 1 |
|  | } |
|  | plotPost(samples[,"theta[1]"], cex.lab = 1.5, xlab=bquote(theta[1]), main=paste("Rel. Freq. Group 1"), |
|  | cred\_mass= x$cred\_mass, col="#5DE293" , show\_median=TRUE, comp\_val=x$comp\_theta[1], xlim=post\_xlim) |
|  | for(i in 2:n\_groups) { |
|  | plotPost(samples[,paste0("theta[",i, "]")], cex.lab = 1.5, xlab=bquote(theta[.(i)]), main=paste("Group", i), |
|  | cred\_mass= x$cred\_mass, col="#5DE293" , show\_median=TRUE, comp\_val=x$comp\_theta[i], xlim=post\_xlim, show\_labels = FALSE) |
|  | } |
|  | diff\_xlim <- range(apply(diff\_samples, 2, quantile, probs = c(0.001, 0.999))) |
|  | if(all(diff\_xlim < 0)) { |
|  | diff\_xlim[2] <- 0 |
|  | } else if(all(diff\_xlim > 0)) { |
|  | diff\_xlim[1] <- 0 |
|  | } |
|  | for(i in 1:ncol(diff\_samples)) { |
|  | diff\_name <- colnames(diff\_samples)[i] |
|  | indices\_match <- str\_match(diff\_name, "\\[(\\d+),(\\d+)\\]$") |
|  | group\_i <- as.numeric(indices\_match[1,2]) |
|  | group\_j <- as.numeric(indices\_match[1,3]) |
|  | plotPost(diff\_samples[,i], cex.lab = 1.5, xlab=bquote(theta[.(group\_i)] - theta[.(group\_j)]), |
|  | main="", cred\_mass= x$cred\_mass, col="skyblue" , show\_median=TRUE, |
|  | comp\_val=0, xlim=diff\_xlim, show\_labels = FALSE) |
|  | plotPost(-diff\_samples[,i], cex.lab = 1.5, xlab=bquote(theta[.(group\_j)] - theta[.(group\_i)]), |
|  | main="", cred\_mass= x$cred\_mass, col="skyblue" , show\_median=TRUE, |
|  | comp\_val=0, xlim=sort(-diff\_xlim), show\_labels = FALSE) |
|  | } |
|  |  |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | print.bayes\_prop\_test <- function(x, ...) { |
|  |  |
|  | s <- format\_stats(x$stats) |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian First Aid proportion test\n") |
|  | cat("\n") |
|  | cat("data: ", x$data\_name, "\n", sep="") |
|  | pad\_width <- max(nchar(as.character(c(x$x, x$n)))) + 1 |
|  | cat("number of successes: ", paste(str\_pad(x$x, pad\_width), collapse = ","), "\n", sep="") |
|  | cat("number of trials: ", paste(str\_pad(x$n, pad\_width), collapse = ","), "\n", sep="") |
|  | cat("Estimated relative frequency of success [", s[1, "HDI%"] ,"% credible interval]:\n", sep="") |
|  | for(param\_i in which(str\_detect(rownames(s), "theta\\["))) { |
|  | param <- paste("theta[", param\_i, "]", sep="") |
|  | cat(" Group ", param\_i,": " ,s[param, "median"], " [", paste(s[param, c("HDIlo", "HDIup")], collapse = ", "),"]\n", sep = "") |
|  | } |
|  |  |
|  | group\_diffs <- format\_group\_diffs(x) |
|  | if(ncol(group\_diffs) > 1) { |
|  | cat("Estimated pairwise group differences (row - column) with", s[1, "HDI%"] ,"% cred. intervals:\n") |
|  | cat(format("Group", width = 2 + nchar(rownames(group\_diffs)[1]) \* 2 + sum(nchar(colnames(group\_diffs))), |
|  | justify = "centre"), "\n", sep="") |
|  | print(format\_group\_diffs(x), quote=FALSE) |
|  | } else { |
|  | cat("Estimated group difference (Group 1 - Group 2):\n") |
|  | cat(" ", str\_trim(group\_diffs[1,1]), " ",group\_diffs[2,1], "\n", sep="") |
|  | } |
|  |  |
|  | if(! is.null(x$comp\_theta)) { |
|  | cat("The prob. that the relative frequency of success is less/more than comp. val:\n") |
|  | comp\_table <- s[str\_detect(rownames(s), "theta\\["), c("comp", "%<comp", "%>comp")] |
|  | rownames(comp\_table) <- paste(" Group ", 1:nrow(comp\_table), ":", sep="") |
|  | colnames(comp\_table) <- c("comp. val.", " <", " >") |
|  | print(format(comp\_table, justify="centre"), quote=FALSE) |
|  | } |
|  | if(ncol(group\_diffs) == 1) { |
|  | cat("The relative frequency of success is larger for Group 1 by a probability\n") |
|  | cat("of", s["theta\_diff[1,2]", "%>comp"], "and larger for Group 2 by a probability of", s["theta\_diff[1,2]", "%<comp"], ".\n") |
|  | } |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_prop\_test <- function(object, ...) { |
|  |  |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | pad\_width <- max(nchar(as.character(c(object$x, object$n)))) + 1 |
|  | cat("number of successes: ", paste(str\_pad(object$x, pad\_width), collapse = ","), "\n", sep="") |
|  | cat("number of trials: ", paste(str\_pad(object$n, pad\_width), collapse = ","), "\n", sep="") |
|  | cat("\n") |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("theta[i]: the relative frequency of success for Group i\n") |
|  | cat("x\_pred[i]: predicted number of successes in a replication for Group i\n") |
|  | cat("theta\_diff[i,j]: the difference between two groups (theta[i] - theta[j])\n") |
|  | cat("\n") |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ," (except for the theta\_diff parameters where\n", sep="") |
|  | cat("the comparison value comp is 0.0).\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_prop\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("theta: The relative frequency of success\n") |
|  | cat("x\_pred: Predicted number of successes in a replication\n") |
|  | cat("theta\_diff[i,j]: the difference between two groups (theta[i] - theta[j])\n") |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.6) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Model code for the Bayesian First Aid alternative to the test of proportions # |
|  |  |
|  | #' @export |
|  | model.code.bayes\_prop\_test <- function(fit) { |
|  | cat("### Model code for the Bayesian First Aid ###\n### alternative to the test of proportions ###\n") |
|  | cat("require(rjags)\n\n") |
|  |  |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", deparse(fit$x, ), "\n") |
|  | cat("n <-", deparse(fit$n), "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(prop\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | prop\_model\_code <- function(x, n) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = list(x = x, n = n), |
|  | n.chains = 3, n.adapt=1000) |
|  | samples <- coda.samples(model, c("theta", "x\_pred"), n.iter=5000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  |  |
|  | # You can extract the mcmc samples as a matrix and compare the thetas |
|  | # of the groups. For example, the following shows the median and 95% |
|  | # credible interval for the difference between Group 1 and Group 2. |
|  | samp\_mat <- as.matrix(samples) |
|  | quantile(samp\_mat[, "theta[1]"] - samp\_mat[, "theta[2]"], c(0.025, 0.5, 0.975)) |
|  | } |
|  | prop\_model\_code <- inject\_model\_string(prop\_model\_code, prop\_model\_string) |

Bayes\_t\_test.R

|  |
| --- |
| bayes.t.test <- function(x, ...) { |
|  | UseMethod("bayes.t.test") |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | #' @rdname bayes.t.test |
|  | bayes.t.test.default <- function(x, y = NULL, alternative = c("two.sided", "less", "greater"), |
|  | mu = 0, paired = FALSE, var.equal = FALSE, cred.mass = 0.95, n.iter = 30000, progress.bar="text", conf.level,...) { |
|  |  |
|  | if(! missing(conf.level)) { |
|  | cred.mass <- conf.level |
|  | } |
|  |  |
|  | if(var.equal) { |
|  | var.equal <- FALSE |
|  | warning("To assume equal variance of 'x' and 'y' is not supported. Continuing by estimating the variance of 'x' and 'y' separately.") |
|  | } |
|  |  |
|  | if(! missing(alternative)) { |
|  | warning("The argument 'alternative' is ignored by bayes.binom.test") |
|  | } |
|  |  |
|  | ### Original (but slighly modified) code from t.test.default ### |
|  | alternative <- match.arg(alternative) |
|  | if (!missing(mu) && (length(mu) != 1 || is.na(mu))) |
|  | stop("'mu' must be a single number") |
|  | if (!missing(cred.mass) && (length(cred.mass) != 1 || !is.finite(cred.mass) || |
|  | cred.mass < 0 || cred.mass > 1)) |
|  | stop("'cred.mass' or 'conf.level' must be a single number between 0 and 1") |
|  |  |
|  | # removing incomplete cases and preparing the data vectors (x & y) |
|  | if (!is.null(y)) { |
|  | x\_name <- deparse(substitute(x)) |
|  | y\_name <- deparse(substitute(y)) |
|  | data\_name <- paste(x\_name, "and", y\_name) |
|  | if (paired) |
|  | xok <- yok <- complete.cases(x, y) |
|  | else { |
|  | yok <- !is.na(y) |
|  | xok <- !is.na(x) |
|  | } |
|  | y <- y[yok] |
|  | } |
|  | else { |
|  | x\_name <- deparse(substitute(x)) |
|  | data\_name <- x\_name |
|  | if (paired) |
|  | stop("'y' is missing for paired test") |
|  | xok <- !is.na(x) |
|  | yok <- NULL |
|  | } |
|  | x <- x[xok] |
|  |  |
|  | # Checking that there is enough data. Even though BEST handles the case with |
|  | # one data point it is still usefull to do these checks. |
|  | nx <- length(x) |
|  | mx <- mean(x) |
|  | vx <- var(x) |
|  | if (is.null(y)) { |
|  | if (nx < 2) |
|  | stop("not enough 'x' observations") |
|  | df <- nx - 1 |
|  | stderr <- sqrt(vx/nx) |
|  | if (stderr < 10 \* .Machine$double.eps \* abs(mx)) |
|  | stop("data are essentially constant") |
|  | } |
|  | else { |
|  | ny <- length(y) |
|  | if (nx < 2) |
|  | stop("not enough 'x' observations") |
|  | if (ny < 2) |
|  | stop("not enough 'y' observations") |
|  | my <- mean(y) |
|  | vy <- var(y) |
|  | stderrx <- sqrt(vx/nx) |
|  | stderry <- sqrt(vy/ny) |
|  | stderr <- sqrt(stderrx^2 + stderry^2) |
|  | df <- stderr^4/(stderrx^4/(nx - 1) + stderry^4/(ny - 1)) |
|  | if (stderr < 10 \* .Machine$double.eps \* max(abs(mx), |
|  | abs(my))) |
|  | stop("data are essentially constant") |
|  | } |
|  |  |
|  | ### Own code starts here ### |
|  |  |
|  | if(paired) { |
|  | mcmc\_samples <- jags\_paired\_t\_test(x, y, n.chains= 3, n.iter = ceiling(n.iter / 3), progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = mu) |
|  | bfa\_object <- list(x = x, y = y, pair\_diff = x - y, comp = mu, cred\_mass = cred.mass, |
|  | x\_name = x\_name, y\_name = y\_name, data\_name = data\_name, |
|  | x\_data\_expr = x\_name, y\_data\_expr = y\_name, |
|  | mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_paired\_t\_test", "bayesian\_first\_aid") |
|  |  |
|  | } else if(is.null(y)) { |
|  | mcmc\_samples <- jags\_one\_sample\_t\_test(x, comp\_mu = mu, n.chains= 3, n.iter = ceiling(n.iter / 3), progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = mu) |
|  | bfa\_object <- list(x = x, comp = mu, cred\_mass = cred.mass, x\_name = x\_name, x\_data\_expr = x\_name, |
|  | data\_name = data\_name, mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_one\_sample\_t\_test", "bayesian\_first\_aid") |
|  |  |
|  | } else { # is two sample t.test |
|  | mcmc\_samples <- jags\_two\_sample\_t\_test(x, y, n.chains= 3, n.iter = ceiling(n.iter / 3), progress.bar=progress.bar) |
|  | stats <- mcmc\_stats(mcmc\_samples, cred\_mass = cred.mass, comp\_val = mu) |
|  | bfa\_object <- list(x = x, y = y, comp = mu, cred\_mass = cred.mass, |
|  | x\_name = x\_name, y\_name = y\_name, data\_name = data\_name, |
|  | x\_data\_expr = x\_name, y\_data\_expr = y\_name, |
|  | mcmc\_samples = mcmc\_samples, stats = stats) |
|  | class(bfa\_object) <- c("bayes\_two\_sample\_t\_test", "bayesian\_first\_aid") |
|  | } |
|  | bfa\_object |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | #' @rdname bayes.t.test |
|  | bayes.t.test.formula <- function(formula, data, subset, na.action, ...) { |
|  |  |
|  | ### Original code from t.test.formula ### |
|  | if (missing(formula) || (length(formula) != 3L) || (length(attr(terms(formula[-2L]), "term.labels")) != 1L)) |
|  | stop("'formula' missing or incorrect") |
|  | m <- match.call(expand.dots = FALSE) |
|  | if (is.matrix(eval(m$data, parent.frame()))) |
|  | m$data <- as.data.frame(data) |
|  | m[[1L]] <- quote(stats::model.frame) |
|  | m$... <- NULL |
|  | mf <- eval(m, parent.frame()) |
|  | data\_name <- paste(names(mf), collapse = " by ") |
|  | response\_name <- names(mf)[1] |
|  | group\_name <- names(mf)[2] |
|  | names(mf) <- NULL |
|  | response <- attr(attr(mf, "terms"), "response") |
|  | g <- factor(mf[[-response]]) |
|  | if (nlevels(g) != 2L) |
|  | stop("grouping factor must have exactly 2 levels") |
|  | DATA <- setNames(split(mf[[response]], g), c("x", "y")) |
|  |  |
|  | ### Own code starts here ### |
|  | bfa\_object <- do.call("bayes.t.test", c(DATA, list(...))) |
|  | bfa\_object$data\_name <- data\_name |
|  | bfa\_object$x\_name <- paste("group", levels(g)[1]) |
|  | bfa\_object$y\_name <- paste("group", levels(g)[2]) |
|  | if(!missing(data)) { |
|  | data\_expr <- deparse(substitute(data)) |
|  | bfa\_object$x\_data\_expr <- |
|  | paste("subset(", data\_expr, ", as.factor(", group\_name, ") == ", |
|  | deparse(levels(g)[1]), ", ", response\_name, ", drop = TRUE)", sep="") |
|  | bfa\_object$y\_data\_expr <- |
|  | paste("subset(", data\_expr, ", as.factor(", group\_name, ") == ", |
|  | deparse(levels(g)[2]), ", ", response\_name, ", drop = TRUE)", sep="") |
|  | } else { |
|  | bfa\_object$x\_data\_expr <- |
|  | paste(response\_name, "[", "as.factor(", group\_name, ") == ", |
|  | deparse(levels(g)[1]),"]",sep="") |
|  | bfa\_object$y\_data\_expr <- |
|  | paste(response\_name, "[", "as.factor(", group\_name, ") == ", |
|  | deparse(levels(g)[2]),"]",sep="") |
|  | } |
|  | bfa\_object |
|  | } |
|  |  |
|  |  |
|  | one\_sample\_t\_model\_string <- "model { |
|  | for(i in 1:length(x)) { |
|  | x[i] ~ dt( mu , tau , nu ) |
|  | } |
|  | x\_pred ~ dt( mu , tau , nu ) |
|  | eff\_size <- (mu - comp\_mu) / sigma |
|  |  |
|  | mu ~ dnorm( mean\_mu , precision\_mu ) |
|  | tau <- 1/pow( sigma , 2 ) |
|  | sigma ~ dunif( sigma\_low , sigma\_high ) |
|  | # A trick to get an exponentially distributed prior on nu that starts at 1. |
|  | nu <- nuMinusOne + 1 |
|  | nuMinusOne ~ dexp(1/29) |
|  | }" |
|  |  |
|  | jags\_one\_sample\_t\_test <- function(x, comp\_mu = 0,n.adapt= 500, n.chains=3, n.update = 100, n.iter=5000, thin=1, progress.bar="text") { |
|  | data\_list <- list( |
|  | x = x, |
|  | mean\_mu = mean(x, trim=0.2) , |
|  | precision\_mu = 1 / (mad0(x)^2 \* 1000000), |
|  | sigma\_low = mad0(x) / 1000 , |
|  | sigma\_high = mad0(x) \* 1000 , |
|  | comp\_mu = comp\_mu |
|  | ) |
|  |  |
|  | inits\_list <- list(mu = mean(x, trim=0.2), sigma = mad0(x), nuMinusOne = 4) |
|  | params <- c("mu", "sigma", "nu", "eff\_size", "x\_pred") |
|  | mcmc\_samples <- run\_jags(one\_sample\_t\_model\_string, data = data\_list, inits = inits\_list, |
|  | params = params, n.chains = n.chains, n.adapt = n.adapt, |
|  | n.update = n.update, n.iter = n.iter, thin = thin, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | two\_sample\_t\_model\_string <- "model { |
|  | for(i in 1:length(x)) { |
|  | x[i] ~ dt( mu\_x , tau\_x , nu ) |
|  | } |
|  | x\_pred ~ dt( mu\_x , tau\_x , nu ) |
|  | for(i in 1:length(y)) { |
|  | y[i] ~ dt( mu\_y , tau\_y , nu ) |
|  | } |
|  | y\_pred ~ dt( mu\_y , tau\_y , nu ) |
|  | eff\_size <- (mu\_x - mu\_y) / sqrt((pow(sigma\_x, 2) + pow(sigma\_y, 2)) / 2) |
|  | mu\_diff <- mu\_x - mu\_y |
|  | sigma\_diff <-sigma\_x - sigma\_y |
|  |  |
|  | # The priors |
|  | mu\_x ~ dnorm( mean\_mu , precision\_mu ) |
|  | tau\_x <- 1/pow( sigma\_x , 2 ) |
|  | sigma\_x ~ dunif( sigma\_low , sigma\_high ) |
|  |  |
|  | mu\_y ~ dnorm( mean\_mu , precision\_mu ) |
|  | tau\_y <- 1/pow( sigma\_y , 2 ) |
|  | sigma\_y ~ dunif( sigma\_low , sigma\_high ) |
|  |  |
|  | # A trick to get an exponentially distributed prior on nu that starts at 1. |
|  | nu <- nuMinusOne+1 |
|  | nuMinusOne ~ dexp(1/29) |
|  | }" |
|  |  |
|  | # Adapted from John Kruschke's original BEST code. |
|  | jags\_two\_sample\_t\_test <- function(x, y, n.adapt= 500, n.chains=3, n.update = 100, n.iter=5000, thin=1, progress.bar="text") { |
|  | data\_list <- list( |
|  | x = x , |
|  | y = y , |
|  | mean\_mu = mean(c(x, y), trim=0.2) , |
|  | precision\_mu = 1 / (mad0(c(x, y))^2 \* 1000000), |
|  | sigma\_low = mad0(c(x, y)) / 1000 , |
|  | sigma\_high = mad0(c(x, y)) \* 1000 |
|  | ) |
|  |  |
|  | inits\_list <- list( |
|  | mu\_x = mean(x, trim=0.2), |
|  | mu\_y = mean(y, trim=0.2), |
|  | sigma\_x = mad0(x), |
|  | sigma\_y = mad0(y), |
|  | nuMinusOne = 4 |
|  | ) |
|  |  |
|  | params <- c("mu\_x", "sigma\_x", "mu\_y", "sigma\_y", "mu\_diff", "sigma\_diff","nu", "eff\_size", "x\_pred", "y\_pred") |
|  | mcmc\_samples <- run\_jags(two\_sample\_t\_model\_string, data = data\_list, inits = inits\_list, |
|  | params = params, n.chains = n.chains, n.adapt = n.adapt, |
|  | n.update = n.update, n.iter = n.iter, thin = thin, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | # Right now, this is basically just calling jags\_one\_sample\_t\_test but I'm |
|  | # keeping it in case I would want to change it in the future. |
|  |  |
|  | paired\_samples\_t\_model\_string <- "model { |
|  | for(i in 1:length(pair\_diff)) { |
|  | pair\_diff[i] ~ dt( mu\_diff , tau\_diff , nu ) |
|  | } |
|  | diff\_pred ~ dt( mu\_diff , tau\_diff , nu ) |
|  | eff\_size <- (mu\_diff - comp\_mu) / sigma\_diff |
|  |  |
|  | mu\_diff ~ dnorm( mean\_mu , precision\_mu ) |
|  | tau\_diff <- 1/pow( sigma\_diff , 2 ) |
|  | sigma\_diff ~ dunif( sigma\_low , sigma\_high ) |
|  | # A trick to get an exponentially distributed prior on nu that starts at 1. |
|  | nu <- nuMinusOne + 1 |
|  | nuMinusOne ~ dexp(1/29) |
|  | }" |
|  |  |
|  |  |
|  | jags\_paired\_t\_test <- function(x, y, comp\_mu = 0, n.adapt= 500, n.chains=3, n.update = 100, n.iter=5000, thin=1, progress.bar="text") { |
|  | pair\_diff <- x - y |
|  | data\_list <- list( |
|  | pair\_diff = pair\_diff, |
|  | mean\_mu = mean(pair\_diff, trim=0.2) , |
|  | precision\_mu = 1 / (mad0(pair\_diff)^2 \* 1000000), |
|  | sigma\_low = mad0(pair\_diff) / 1000 , |
|  | sigma\_high = mad0(pair\_diff) \* 1000 , |
|  | comp\_mu = comp\_mu |
|  | ) |
|  |  |
|  | inits\_list <- list(mu\_diff = mean(pair\_diff, trim=0.2), |
|  | sigma\_diff = mad0(pair\_diff), |
|  | nuMinusOne = 4) |
|  | params <- c("mu\_diff", "sigma\_diff", "nu", "eff\_size", "diff\_pred") |
|  | mcmc\_samples <- run\_jags(paired\_samples\_t\_model\_string, data = data\_list, inits = inits\_list, |
|  | params = params, n.chains = n.chains, n.adapt = n.adapt, |
|  | n.update = n.update, n.iter = n.iter, thin = thin, progress.bar=progress.bar) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | #################################### |
|  | ### One sample t-test S3 methods ### |
|  | #################################### |
|  |  |
|  | #' @export |
|  | print.bayes\_one\_sample\_t\_test <- function(x, ...) { |
|  |  |
|  | s <- format\_stats(x$stats) |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian estimation supersedes the t test (BEST) - one sample\n") |
|  | cat("\n") |
|  | cat("data: ", x$data\_name, ", n = ", length(x$x),"\n", sep="") |
|  | cat("\n") |
|  | cat(" Estimates [", s[1, "HDI%"] ,"% credible interval]\n", sep="") |
|  | cat("mean of ", x$x\_name, ": ", s["mu", "median"], " [", s["mu", "HDIlo"], ", ", s["mu", "HDIup"] , "]\n",sep="") |
|  | cat("sd of ", x$x\_name, ": ", s["sigma", "median"], " [", s["sigma", "HDIlo"], ", ", s["sigma", "HDIup"] , "]\n",sep="") |
|  |  |
|  | cat("\n") |
|  | cat("The mean is more than", s["mu","comp"] , "by a probability of", s["mu","%>comp"], "\n") |
|  | cat("and less than", s["mu", "comp"] , "by a probability of", s["mu", "%<comp"], "\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | print\_bayes\_one\_sample\_t\_test\_params <- function(x) { |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("mu: the mean of", x$data\_name, "\n") |
|  | cat("sigma: the scale of", x$data\_name,", a consistent\n estimate of SD when nu is large.\n") |
|  | cat("nu: the degrees-of-freedom for the t distribution fitted to",x$data\_name , "\n") |
|  | cat("eff\_size: the effect size calculated as (mu - ", x$comp ,") / sigma\n", sep="") |
|  | cat("x\_pred: predicted distribution for a new datapoint generated as",x$data\_name , "\n") |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_one\_sample\_t\_test <- function(object, ...) { |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | cat(object$data\_name, ", n = ", length(object$x), "\n", sep="") |
|  | cat("\n") |
|  |  |
|  | print\_bayes\_one\_sample\_t\_test\_params(object) |
|  | cat("\n") |
|  |  |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_one\_sample\_t\_test |
|  | #' @export |
|  | plot.bayes\_one\_sample\_t\_test <- function(x, ...) { |
|  | stats <- x$stats |
|  | mcmc\_samples <- x$mcmc\_samples |
|  | samples\_mat <- as.matrix(mcmc\_samples) |
|  | mu = samples\_mat[,"mu"] |
|  | sigma = samples\_mat[,"sigma"] |
|  | nu = samples\_mat[,"nu"] |
|  |  |
|  | old\_par <- par(no.readonly = TRUE) |
|  | #layout( matrix( c(3,3,4,4,5,5, 1,1,1,1,2,2) , nrow=6, ncol=2 , byrow=FALSE ) ) |
|  | layout( matrix( c(2,2,3,3,4,4, 1,1) , nrow=4, ncol=2 , byrow=FALSE ) ) |
|  | par( mar=c(3.5,3.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  |  |
|  | n\_curves <- 30 |
|  | rand\_i <- sample(nrow(samples\_mat), n\_curves) |
|  | hist\_with\_t\_curves(x$x, stats["mu", "mean"], stats["sigma", "median"], mu[rand\_i], sigma[rand\_i], |
|  | nu[rand\_i], x$data\_name, main= "Data w. Post. Pred.", x\_range= range(x$x)) |
|  |  |
|  | # Plot posterior distribution of parameter nu: |
|  | #paramInfo = plotPost( log10(nu) , col="skyblue" , |
|  | # xlab=bquote("log10("\*nu\*")") , cex.lab = 1.75 , show\_mode=TRUE , |
|  | # main="Normality" ) # (<0.7 suggests kurtosis) |
|  |  |
|  | # distribution of mu: |
|  | xlim = range( c( mu , x$comp ) ) |
|  | plotPost(mu , xlim=xlim , cex.lab = 1.75 , comp\_val = x$comp, cred\_mass= x$cred\_mass, |
|  | xlab=bquote(mu) , main=paste("Mean") , col="skyblue", show\_median=TRUE ) |
|  |  |
|  | # distribution of sigma: |
|  | plotPost(sigma, cex.lab = 1.75, xlab=bquote(sigma), main=paste("Std. Dev."), |
|  | cred\_mass= x$cred\_mass, col="skyblue" , show\_median=TRUE ) |
|  |  |
|  | # effect size: |
|  | plotPost(samples\_mat[, "eff\_size"] , comp\_val=0 , xlab=bquote( (mu-.(x$comp)) / sigma ), |
|  | cred\_mass= x$cred\_mass, show\_median=TRUE , cex.lab=1.75 , main="Effect Size" , col="skyblue" ) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_one\_sample\_t\_test <- function(fit) { |
|  |  |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | print\_bayes\_one\_sample\_t\_test\_params(fit) |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_one\_sample\_t\_test <- function(fit) { |
|  | cat("### Model code for Bayesian estimation supersedes the t test - one sample ###\n") |
|  |  |
|  | cat("require(rjags)\n\n") |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x\_data\_expr, "\n") |
|  | cat("comp\_mu <- ", fit$comp, "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(one\_sample\_t\_test\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | one\_sample\_t\_test\_model\_code <- function(x, comp\_mu) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Setting parameters for the priors that in practice will result |
|  | # in flat priors on mu and sigma. |
|  | mean\_mu = mean(x, trim=0.2) |
|  | precision\_mu = 1 / (mad(x)^2 \* 1000000) |
|  | sigma\_low = mad(x) / 1000 |
|  | sigma\_high = mad(x) \* 1000 |
|  |  |
|  | # Initializing parameters to sensible starting values helps the convergence |
|  | # of the MCMC sampling. Here using robust estimates of the mean (trimmed) |
|  | # and standard deviation (MAD). |
|  | inits\_list <- list(mu = mean(x, trim=0.2), sigma = mad(x), nuMinusOne = 4) |
|  |  |
|  | data\_list <- list( |
|  | x = x, |
|  | comp\_mu = comp\_mu, |
|  | mean\_mu = mean\_mu, |
|  | precision\_mu = precision\_mu, |
|  | sigma\_low = sigma\_low, |
|  | sigma\_high = sigma\_high) |
|  |  |
|  | # The parameters to monitor. |
|  | params <- c("mu", "sigma", "nu", "eff\_size", "x\_pred") |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = data\_list, |
|  | inits = inits\_list, n.chains = 3, n.adapt=1000) |
|  | update(model, 500) # Burning some samples to the MCMC gods.... |
|  | samples <- coda.samples(model, params, n.iter=10000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | one\_sample\_t\_test\_model\_code <- inject\_model\_string(one\_sample\_t\_test\_model\_code, one\_sample\_t\_model\_string) |
|  |  |
|  |  |
|  |  |
|  | #################################### |
|  | ### Two sample t-test S3 methods ### |
|  | #################################### |
|  | #' @export |
|  | print.bayes\_two\_sample\_t\_test <- function(x, ...) { |
|  | s <- format\_stats(x$stats) |
|  |  |
|  | cat("\n") |
|  | cat("\tBayesian estimation supersedes the t test (BEST) - two sample\n") |
|  | cat("\n") |
|  | cat("data: ", x$x\_name, " (n = ", length(x$x) ,") and ", x$y\_name," (n = ", length(x$y) ,")\n", sep="") |
|  | cat("\n") |
|  | cat(" Estimates [", s[1, "HDI%"] ,"% credible interval]\n", sep="") |
|  | cat("mean of ", x$x\_name, ": ", s["mu\_x", "median"], " [", s["mu\_x", "HDIlo"], ", ", s["mu\_x", "HDIup"] , "]\n",sep="") |
|  | cat("mean of ", x$y\_name, ": ", s["mu\_y", "median"], " [", s["mu\_y", "HDIlo"], ", ", s["mu\_y", "HDIup"] , "]\n",sep="") |
|  | cat("difference of the means: ", s["mu\_diff", "median"], " [", s["mu\_diff", "HDIlo"], ", ", s["mu\_diff", "HDIup"] , "]\n",sep="") |
|  | cat("sd of ", x$x\_name, ": ", s["sigma\_x", "median"], " [", s["sigma\_x", "HDIlo"], ", ", s["sigma\_x", "HDIup"] , "]\n",sep="") |
|  | cat("sd of ", x$y\_name, ": ", s["sigma\_y", "median"], " [", s["sigma\_y", "HDIlo"], ", ", s["sigma\_y", "HDIup"] , "]\n",sep="") |
|  |  |
|  | cat("\n") |
|  | cat("The difference of the means is greater than", s["mu\_diff","comp"] , "by a probability of", s["mu\_diff","%>comp"], "\n") |
|  | cat("and less than", s["mu\_diff", "comp"] , "by a probability of", s["mu\_diff", "%<comp"], "\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  |  |
|  | print\_bayes\_two\_sample\_t\_test\_params <- function(x) { |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("mu\_x: the mean of", x$x\_name, "\n") |
|  | cat("sigma\_x: the scale of", x$x\_name,", a consistent\n estimate of SD when nu is large.\n") |
|  | cat("mu\_y: the mean of", x$y\_name, "\n") |
|  | cat("sigma\_y: the scale of", x$y\_name,"\n") |
|  | cat("mu\_diff: the difference in means (mu\_x - mu\_y)\n") |
|  | cat("sigma\_diff: the difference in scale (sigma\_x - sigma\_y)\n") |
|  | cat("nu: the degrees-of-freedom for the t distribution\n") |
|  | cat(" fitted to",x$data\_name , "\n") |
|  | cat("eff\_size: the effect size calculated as \n", sep="") |
|  | cat(" (mu\_x - mu\_y) / sqrt((sigma\_x^2 + sigma\_y^2) / 2)\n", sep="") |
|  | cat("x\_pred: predicted distribution for a new datapoint\n") |
|  | cat(" generated as",x$x\_name , "\n") |
|  | cat("y\_pred: predicted distribution for a new datapoint\n") |
|  | cat(" generated as",x$y\_name , "\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_two\_sample\_t\_test <- function(object, ...) { |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | cat(object$x\_name, ", n = ", length(object$x), "\n", sep="") |
|  | cat(object$y\_name, ", n = ", length(object$y), "\n", sep="") |
|  | cat("\n") |
|  |  |
|  | print\_bayes\_two\_sample\_t\_test\_params(object) |
|  | cat("\n") |
|  |  |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_two\_sample\_t\_test |
|  | #' @export |
|  | plot.bayes\_two\_sample\_t\_test <- function(x, ...) { |
|  | stats <- x$stats |
|  | mcmc\_samples <- x$mcmc\_samples |
|  | samples\_mat <- as.matrix(mcmc\_samples) |
|  | mu\_x = samples\_mat[,"mu\_x"] |
|  | sigma\_x = samples\_mat[,"sigma\_x"] |
|  | mu\_y = samples\_mat[,"mu\_y"] |
|  | sigma\_y = samples\_mat[,"sigma\_y"] |
|  | nu = samples\_mat[,"nu"] |
|  |  |
|  | old\_par <- par(no.readonly = TRUE) |
|  | #layout( matrix( c(4,5,7,8,3,1,2,6,9,10) , nrow=5, byrow=FALSE ) ) |
|  | layout( matrix( c(3,4,5,1,2,6) , nrow=3, byrow=FALSE ) ) |
|  | par( mar=c(3.5,3.5,2.5,0.51) , mgp=c(2.25,0.7,0) ) |
|  |  |
|  |  |
|  | # Plot data with post predictive distribution |
|  | n\_curves <- 30 |
|  | data\_range <- range(c(x$x, x$y)) |
|  | rand\_i <- sample(nrow(samples\_mat), n\_curves) |
|  | hist\_with\_t\_curves(x$x, stats["mu\_x", "mean"], stats["sigma\_x", "median"], mu\_x[rand\_i], sigma\_x[rand\_i], |
|  | nu[rand\_i], x$x\_name, main= paste("Data", x$x\_name, "w. Post. Pred."), x\_range= data\_range) |
|  | hist\_with\_t\_curves(x$y, stats["mu\_y", "mean"], stats["sigma\_y", "median"], mu\_y[rand\_i], sigma\_y[rand\_i], |
|  | nu[rand\_i], x$y\_name, main= paste("Data", x$y\_name, "w. Post. Pred."), x\_range= data\_range) |
|  |  |
|  | # Plot posterior distribution of parameter nu: |
|  | # plotPost( log10(nu) , col="skyblue" , cred\_mass= x$cred\_mass, |
|  | # xlab=bquote("log10("\*nu\*")") , cex.lab = 1.75 , show\_mode=TRUE , |
|  | # main="Normality" ) # (<0.7 suggests kurtosis) |
|  |  |
|  | # Plot posterior distribution of parameters mu\_x, mu\_y, and their difference: |
|  | xlim = range( c( mu\_x , mu\_y ) ) |
|  | plotPost( mu\_x , xlim=xlim , cex.lab = 1.75 , cred\_mass= x$cred\_mass, show\_median=TRUE, |
|  | xlab=bquote(mu[x]) , main=paste(x$x\_name,"Mean") , col="skyblue" ) |
|  | plotPost( mu\_y , xlim=xlim , cex.lab = 1.75 , cred\_mass= x$cred\_mass, show\_median=TRUE, |
|  | xlab=bquote(mu[y]) , main=paste(x$y\_name,"Mean") , col="skyblue" ) |
|  | plotPost( samples\_mat[,"mu\_diff"] , comp\_val= x$comp , cred\_mass= x$cred\_mass, |
|  | xlab=bquote(mu[x] - mu[y]) , cex.lab = 1.75 , show\_median=TRUE, |
|  | main="Difference of Means" , col="skyblue" ) |
|  |  |
|  | # Save this to var.test |
|  | # |
|  | # Plot posterior distribution of param's sigma\_x, sigma\_y, and their difference: |
|  | # xlim=range( c( sigma\_x , sigma\_y ) ) |
|  | # plotPost( sigma\_x , xlim=xlim , cex.lab = 1.75 ,cred\_mass= x$cred\_mass, |
|  | # xlab=bquote(sigma[x]) , main=paste(x$x\_name, "Std. Dev.") , |
|  | # col="skyblue" , show\_mode=TRUE ) |
|  | # plotPost( sigma\_y , xlim=xlim , cex.lab = 1.75 ,cred\_mass= x$cred\_mass, |
|  | # xlab=bquote(sigma[y]) , main=paste(x$y\_name, "Std. Dev.") , |
|  | # col="skyblue" , show\_mode=TRUE ) |
|  | # plotPost( samples\_mat[, "sigma\_diff"] , comp\_val= x$comp , cred\_mass= x$cred\_mass, |
|  | # xlab=bquote(sigma[x] - sigma[y]) , cex.lab = 1.75 , |
|  | # main="Difference of Std. Dev.s" , col="skyblue" , show\_mode=TRUE ) |
|  |  |
|  | # Plot of estimated effect size. Effect size is d-sub-a from |
|  | # Macmillan & Creelman, 1991; Simpson & Fitter, 1973; Swets, 1986a, 1986b. |
|  | plotPost( samples\_mat[, "eff\_size"] , comp\_val=0 , cred\_mass= x$cred\_mass, |
|  | xlab=bquote( (mu[x]-mu[y]) / sqrt((sigma[x]^2 +sigma[y]^2 )/2 ) ), |
|  | show\_median=TRUE , cex.lab=1.0 , main="Effect Size" , col="skyblue" ) |
|  |  |
|  | par(old\_par) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_two\_sample\_t\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | print\_bayes\_two\_sample\_t\_test\_params(fit) |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_two\_sample\_t\_test <- function(fit) { |
|  | cat("## Model code for Bayesian estimation supersedes the t test - two sample ##\n") |
|  |  |
|  | cat("require(rjags)\n\n") |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x\_data\_expr, "\n") |
|  | cat("y <-", fit$y\_data\_expr, "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(two\_sample\_t\_test\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | two\_sample\_t\_test\_model\_code <- function(x, y) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Setting parameters for the priors that in practice will result |
|  | # in flat priors on the mu's and sigma's. |
|  | mean\_mu = mean( c(x, y), trim=0.2) |
|  | precision\_mu = 1 / (mad( c(x, y) )^2 \* 1000000) |
|  | sigma\_low = mad( c(x, y) ) / 1000 |
|  | sigma\_high = mad( c(x, y) ) \* 1000 |
|  |  |
|  | # Initializing parameters to sensible starting values helps the convergence |
|  | # of the MCMC sampling. Here using robust estimates of the mean (trimmed) |
|  | # and standard deviation (MAD). |
|  | inits\_list <- list( |
|  | mu\_x = mean(x, trim=0.2), mu\_y = mean(y, trim=0.2), |
|  | sigma\_x = mad(x), sigma\_y = mad(y), |
|  | nuMinusOne = 4) |
|  |  |
|  | data\_list <- list( |
|  | x = x, y = y, |
|  | mean\_mu = mean\_mu, |
|  | precision\_mu = precision\_mu, |
|  | sigma\_low = sigma\_low, |
|  | sigma\_high = sigma\_high) |
|  |  |
|  | # The parameters to monitor. |
|  | params <- c("mu\_x", "mu\_y", "mu\_diff", "sigma\_x", "sigma\_y", "sigma\_diff", |
|  | "nu", "eff\_size", "x\_pred", "y\_pred") |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = data\_list, |
|  | inits = inits\_list, n.chains = 3, n.adapt=1000) |
|  | update(model, 500) # Burning some samples to the MCMC gods.... |
|  | samples <- coda.samples(model, params, n.iter=10000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | two\_sample\_t\_test\_model\_code <- inject\_model\_string(two\_sample\_t\_test\_model\_code, two\_sample\_t\_model\_string) |
|  |  |
|  | ######################################## |
|  | ### Paired samples t-test S3 methods ### |
|  | ######################################## |
|  |  |
|  |  |
|  | #' @export |
|  | print.bayes\_paired\_t\_test <- function(x, ...) { |
|  | s <- format\_stats(x$stats) |
|  | cat("\n") |
|  | cat("\tBayesian estimation supersedes the t test (BEST) - paired samples\n") |
|  | cat("\n") |
|  | cat("data: ", x$data\_name, ", n = ", length(x$pair\_diff),"\n", sep="") |
|  | cat("\n") |
|  | cat(" Estimates [", s[1, "HDI%"] ,"% credible interval]\n", sep="") |
|  | cat("mean paired difference: ", s["mu\_diff", "median"], " [", s["mu\_diff", "HDIlo"], ", ", s["mu\_diff", "HDIup"] , "]\n",sep="") |
|  | cat("sd of the paired differences: ", s["sigma\_diff", "median"], " [", s["sigma\_diff", "HDIlo"], ", ", s["sigma\_diff", "HDIup"] , "]\n",sep="") |
|  |  |
|  | cat("\n") |
|  | cat("The mean difference is more than", s["mu\_diff","comp"] , "by a probability of", s["mu\_diff","%>comp"], "\n") |
|  | cat("and less than", s["mu\_diff", "comp"] , "by a probability of", s["mu\_diff", "%<comp"], "\n") |
|  | cat("\n") |
|  | invisible(NULL) |
|  | } |
|  |  |
|  |  |
|  | print\_bayes\_paired\_t\_test\_params <- function(x) { |
|  |  |
|  | cat(" Model parameters and generated quantities\n") |
|  | cat("mu\_diff: the mean pairwise difference between", x$x\_name, "and", x$y\_name, "\n") |
|  | cat("sigma\_diff: the scale of the pairwise difference, a consistent\n estimate of SD when nu is large.\n") |
|  | cat("nu: the degrees-of-freedom for the t distribution fitted to the pairwise difference\n") |
|  | cat("eff\_size: the effect size calculated as (mu\_diff - ", x$comp ,") / sigma\_diff\n", sep="") |
|  | cat("diff\_pred: predicted distribution for a new datapoint generated\n as the pairwise difference between", x$x\_name, "and", x$y\_name,"\n") |
|  | } |
|  |  |
|  | #' @export |
|  | summary.bayes\_paired\_t\_test <- function(object, ...) { |
|  | s <- round(object$stats, 3) |
|  |  |
|  | cat(" Data\n") |
|  | cat(object$x\_name, ", n = ", length(object$x), "\n", sep="") |
|  | cat(object$y\_name, ", n = ", length(object$y), "\n", sep="") |
|  | cat("\n") |
|  |  |
|  | print\_bayes\_paired\_t\_test\_params(object) |
|  | cat("\n") |
|  |  |
|  | cat(" Measures\n" ) |
|  | print(s[, c("mean", "sd", "HDIlo", "HDIup", "%<comp", "%>comp")]) |
|  | cat("\n") |
|  | cat("'HDIlo' and 'HDIup' are the limits of a ", s[1, "HDI%"] ,"% HDI credible interval.\n", sep="") |
|  | cat("'%<comp' and '%>comp' are the probabilities of the respective parameter being\n") |
|  | cat("smaller or larger than ", s[1, "comp"] ,".\n", sep="") |
|  |  |
|  | cat("\n") |
|  | cat(" Quantiles\n" ) |
|  | print(s[, c("q2.5%", "q25%", "median","q75%", "q97.5%")] ) |
|  | invisible(object$stats) |
|  | } |
|  |  |
|  | #' @method plot bayes\_paired\_t\_test |
|  | #' @export |
|  | plot.bayes\_paired\_t\_test <- function(x, y, ...) { |
|  | stats <- x$stats |
|  | mcmc\_samples <- x$mcmc\_samples |
|  | samples\_mat <- as.matrix(mcmc\_samples) |
|  | mu\_diff = samples\_mat[,"mu\_diff"] |
|  | sigma\_diff = samples\_mat[,"sigma\_diff"] |
|  | nu = samples\_mat[,"nu"] |
|  |  |
|  | old\_par <- par(no.readonly = TRUE) |
|  | #layout( matrix( c(3,3,4,4,5,5, 1,1,1,1,2,2) , nrow=6, ncol=2 , byrow=FALSE ) ) |
|  | layout( matrix( c(2,2,3,3,4,4, 1,1) , nrow=4, ncol=2 , byrow=FALSE ) ) |
|  | par( mar=c(3.5,3.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  |  |
|  | n\_curves <- 30 |
|  | rand\_i <- sample(nrow(samples\_mat), n\_curves) |
|  | hist\_with\_t\_curves(x$pair\_diff, stats["mu\_diff", "mean"], stats["sigma\_diff", "median"], mu\_diff[rand\_i], sigma\_diff[rand\_i], |
|  | nu[rand\_i], x$data\_name, main= "Data w. Post. Pred.", x\_range= range(x$pair\_diff)) |
|  |  |
|  | # Plot posterior distribution of parameter nu: |
|  | #paramInfo = plotPost( log10(nu) , col="skyblue" , |
|  | # xlab=bquote("log10("\*nu\*")") , cex.lab = 1.75 , show\_mode=TRUE , |
|  | # main="Normality" ) # (<0.7 suggests kurtosis) |
|  |  |
|  | # distribution of mu\_diff: |
|  | xlim = range( c( mu\_diff , x$comp ) ) |
|  | plotPost(mu\_diff , xlim=xlim , cex.lab = 1.75 , comp\_val = x$comp, cred\_mass= x$cred\_mass, |
|  | xlab=bquote(mu[diff]) , main=paste("Mean difference") , col="skyblue", show\_median=TRUE ) |
|  |  |
|  | # distribution of sigma\_diff: |
|  | plotPost(sigma\_diff, cex.lab = 1.75, xlab=bquote(sigma[diff]), main=paste("Std. Dev. of difference"), |
|  | cred\_mass= x$cred\_mass, col="skyblue" , show\_median=TRUE ) |
|  |  |
|  | # effect size: |
|  | plotPost(samples\_mat[, "eff\_size"] , comp\_val=0 , xlab=bquote( (mu[diff] -.(x$comp)) / sigma[diff] ), |
|  | cred\_mass= x$cred\_mass, show\_median=TRUE , cex.lab=1.75 , main="Effect Size" , col="skyblue" ) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | diagnostics.bayes\_paired\_t\_test <- function(fit) { |
|  | print\_mcmc\_info(fit$mcmc\_samples) |
|  | cat("\n") |
|  | print\_diagnostics\_measures(round(fit$stats, 3)) |
|  | cat("\n") |
|  | print\_bayes\_paired\_t\_test\_params(fit) |
|  | cat("\n") |
|  |  |
|  | old\_par <- par( mar=c(3.5,2.5,2.5,0.5) , mgp=c(2.25,0.7,0) ) |
|  | plot(fit$mcmc\_samples) |
|  | par(old\_par) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | #' @export |
|  | model.code.bayes\_paired\_t\_test <- function(fit) { |
|  | cat("## Model code for Bayesian estimation supersedes the t test - paired samples ##\n") |
|  |  |
|  | cat("require(rjags)\n\n") |
|  | cat("# Setting up the data\n") |
|  | cat("x <-", fit$x\_data\_expr, "\n") |
|  | cat("y <-", fit$y\_data\_expr, "\n") |
|  | cat("pair\_diff <- x - y\n") |
|  | cat("comp\_mu <- ", fit$comp, "\n") |
|  | cat("\n") |
|  | pretty\_print\_function\_body(paired\_samples\_t\_test\_model\_code) |
|  | invisible(NULL) |
|  | } |
|  |  |
|  | # Not to be run, just to be printed |
|  | paired\_samples\_t\_test\_model\_code <- function(pair\_diff, comp\_mu) { |
|  | # The model string written in the JAGS language |
|  | BayesianFirstAid::replace\_this\_with\_model\_string |
|  |  |
|  | # Setting parameters for the priors that in practice will result |
|  | # in flat priors on mu and sigma. |
|  | mean\_mu = mean(pair\_diff, trim=0.2) |
|  | precision\_mu = 1 / (mad(pair\_diff)^2 \* 1000000) |
|  | sigma\_low = mad(pair\_diff) / 1000 |
|  | sigma\_high = mad(pair\_diff) \* 1000 |
|  |  |
|  | # Initializing parameters to sensible starting values helps the convergence |
|  | # of the MCMC sampling. Here using robust estimates of the mean (trimmed) |
|  | # and standard deviation (MAD). |
|  | inits\_list <- list( |
|  | mu\_diff = mean(pair\_diff, trim=0.2), |
|  | sigma\_diff = mad(pair\_diff), |
|  | nuMinusOne = 4) |
|  |  |
|  | data\_list <- list( |
|  | pair\_diff = pair\_diff, |
|  | comp\_mu = comp\_mu, |
|  | mean\_mu = mean\_mu, |
|  | precision\_mu = precision\_mu, |
|  | sigma\_low = sigma\_low, |
|  | sigma\_high = sigma\_high) |
|  |  |
|  | # The parameters to monitor. |
|  | params <- c("mu\_diff", "sigma\_diff", "nu", "eff\_size", "diff\_pred") |
|  |  |
|  | # Running the model |
|  | model <- jags.model(textConnection(model\_string), data = data\_list, |
|  | inits = inits\_list, n.chains = 3, n.adapt=1000) |
|  | update(model, 500) # Burning some samples to the MCMC gods.... |
|  | samples <- coda.samples(model, params, n.iter=10000) |
|  |  |
|  | # Inspecting the posterior |
|  | plot(samples) |
|  | summary(samples) |
|  | } |
|  | paired\_samples\_t\_test\_model\_code <- inject\_model\_string(paired\_samples\_t\_test\_model\_code, paired\_samples\_t\_model\_string) |

Utility\_fuctions.R

|  |
| --- |
| run\_jags <- function(model\_string, data, inits, params, n.chains, n.adapt, n.update, n.iter, thin, progress.bar) { |
|  | if(!interactive()) { |
|  | progress.bar <- "none" |
|  | } |
|  |  |
|  | # Set the random number generator and seed based on R's random state (through runif) |
|  | if(is.null(inits$.RNG.seed) & is.null(inits$.RNG.name)) { |
|  | RNGs <- c("base::Wichmann-Hill", "base::Marsaglia-Multicarry", |
|  | "base::Super-Duper", "base::Mersenne-Twister") |
|  | init\_list <- inits |
|  | inits <- function(chain) { |
|  | chain\_init\_list <- init\_list |
|  | chain\_init\_list$.RNG.seed <- as.integer(runif(1, 0, .Machine$integer.max)) |
|  | chain\_init\_list$.RNG.name <- RNGs[ ((chain - 1) %% 4) + 1 ] |
|  | chain\_init\_list |
|  | } |
|  | } |
|  |  |
|  | jags\_model <- jags.model(textConnection(model\_string) , data=data , inits=inits , |
|  | n.chains=n.chains , n.adapt=0, quiet=TRUE) |
|  | adapt(jags\_model, max(1, n.adapt), progress.bar="none", end.adaptation=TRUE) |
|  | if(n.update > 0) { |
|  | update( jags\_model, n.update, progress.bar="none") |
|  | } |
|  | mcmc\_samples <- coda.samples( jags\_model , variable.names= params, |
|  | n.iter=n.iter, thin=thin, progress.bar=progress.bar) |
|  | mcmc\_samples <- reorder\_coda(mcmc\_samples, params) |
|  | mcmc\_samples |
|  | } |
|  |  |
|  | # The following two functions are hacks that are only used to construct |
|  | # and print the model code that is printed by the model.code functions. |
|  |  |
|  | # This first function takes a function and replaces the placeholder |
|  | # with model\_string. Is used so that model string doesn't have to be written twice. |
|  | inject\_model\_string <- function(fn, model\_string, placeholder = "BayesianFirstAid::replace\_this\_with\_model\_string") { |
|  | code\_string <- deparse(fn, control="useSource") |
|  | code\_string <- gsub(placeholder, |
|  | paste0('model\_string <- "', gsub("\n", "\n ", model\_string), '"'), code\_string) |
|  | eval(parse(text=code\_string)) |
|  | } |
|  |  |
|  | # This second function pretty prints the body of a function. |
|  | pretty\_print\_function\_body <- function(fn) { |
|  | fn\_string <- deparse(fn, control="useSource") |
|  | fn\_string <- gsub("^ ", "", fn\_string) |
|  | cat(paste(fn\_string[-c(1, length(fn\_string))], collapse="\n")) |
|  | } |
|  |  |
|  | # mcmc\_samples should be a coda mcmc object |
|  | print\_mcmc\_info <- function(mcmc\_samples) { |
|  | cat("\n", "Iterations = ", start(mcmc\_samples), ":", end(mcmc\_samples), "\n", sep = "") |
|  | cat("Thinning interval =", thin(mcmc\_samples), "\n") |
|  | cat("Number of chains =", nchain(mcmc\_samples), "\n") |
|  | cat("Sample size per chain =", (end(mcmc\_samples) - start(mcmc\_samples))/thin(mcmc\_samples) + 1, "\n") |
|  | } |
|  |  |
|  | # s is the matrix of statistics generated by the mcmc\_stats function |
|  | print\_diagnostics\_measures <- function(s) { |
|  | cat(" Diagnostic measures\n") |
|  | print(s[, c("mean", "sd", "mcmc\_se", "n\_eff", "Rhat")]) |
|  | cat("\n") |
|  | cat("mcmc\_se: the estimated standard error of the MCMC approximation of the mean.\n") |
|  | cat("n\_eff: a crude measure of effective MCMC sample size.\n") |
|  | cat("Rhat: the potential scale reduction factor (at convergence, Rhat=1).\n") |
|  | } |
|  |  |
|  | # Below shold be same as: |
|  | #rmt(100, samples\_mat[i, c("mu[1]", "mu[2]")], cov\_mat, samples\_mat[i, "nu"]) |
|  |  |
|  | rmt <- function(n, mu, cov\_mat, nu) { |
|  | t(t(mvrnorm(n, rep(0, length(mu)), cov\_mat) / sqrt(nu / rchisq(n, nu))) + mu) |
|  | } |
|  |  |
|  | # A version of mad (median absolute deviation), that guards against mad returning zero by |
|  | # replacing 0 it by the SD instead. Also removes NAs by default. |
|  |  |
|  | mad0 <- function(..., na.rm=TRUE) { |
|  | mad\_est <- mad(..., na.rm=na.rm) |
|  | if(mad\_est != 0) { |
|  | mad\_est |
|  | } else { |
|  | sd(..., na.rm=na.rm) |
|  | } |
|  | } |
|  |  |
|  | # Estimate the mode of a continous distribution given by x |
|  | # from and to limits the location of the mode |
|  | estimate\_mode <- function(x,from=min(x), to=max(x)) { |
|  | d <- density(x, from=from, to=to) |
|  | d$x[which.max(d$y)] |
|  | } |
|  |  |
|  | # Not very general function that is made to plot a histogram given discrete (integer) |
|  | # data. |
|  | discrete\_hist <- function(x, xlim, col="skyblue", lwd=3, x\_marked = c(), marked\_col = "red", yaxt="n",...) { |
|  | hist\_data <- hist(x, (xlim[1] - 1):(xlim[2]) + 0.5 , plot=FALSE) |
|  | cols <- ifelse(hist\_data$mids %in% x\_marked, rgb(0, 0, 0, 0), col ) |
|  | plot(hist\_data$mids, hist\_data$density, type="h", col=col, lwd=lwd, bty = "o", lend=1,...) |
|  | points(hist\_data$mids[hist\_data$mids %in% x\_marked], hist\_data$density[hist\_data$mids %in% x\_marked], |
|  | type="h", col=marked\_col, lwd=lwd, bty = "o",lend=2,...) |
|  | invisible(hist\_data) |
|  | } |
|  |  |
|  | # Plots a histogram of x with curves of t distributions |
|  | # specified by mu, sigma and nu |
|  | # data\_mu and data\_sigma should be point estimates |
|  | # Adapted from Kruschkes BEST1Gplot |
|  | hist\_with\_t\_curves <- function(data, data\_mu, data\_sigma, mu, sigma, nu, data\_name = "", main = "", |
|  | x\_range = range(data), horiz=FALSE, plot\_n = TRUE, x\_lim=NULL, axs= "r",...) { |
|  | n\_curves <- length(mu) |
|  |  |
|  | if(is.null(x\_lim)) { |
|  | # Calculating limits for the curves |
|  | x\_lim = c( x\_range[1]-0.1\*(x\_range[2]-x\_range[1]) , |
|  | x\_range[2]+0.1\*(x\_range[2]-x\_range[1]) ) |
|  | } |
|  | x = seq( x\_lim[1] , x\_lim[2] , length=200 ) |
|  |  |
|  | # Limits and bins for the histogram |
|  | bin\_with = data\_sigma/2 |
|  | hist\_center = data\_mu |
|  | breaks = sort( c( seq( hist\_center - bin\_with/2 , min(x) - bin\_with/2 , |
|  | -bin\_with ), |
|  | seq( hist\_center + bin\_with/2 , max(x) + bin\_with/2 , |
|  | bin\_with ) , x\_lim ) ) |
|  | hist\_info = hist( data , plot=FALSE , breaks=breaks ) |
|  | hist\_y = hist\_info$density |
|  | hist\_y[ hist\_y==0.0 ] = NA |
|  | hist\_x = hist\_info$mids |
|  | hist\_x[ hist\_y==0.0 ] = NA |
|  |  |
|  | # The height of the plot |
|  | curve\_maxs <- sapply(seq\_along(mu), function(i) { |
|  | max(dt( (x - mu[i]) / sigma[i] , df=nu[i] ) / sigma[i]) |
|  | }) |
|  |  |
|  | max\_y = max( curve\_maxs, max(hist\_y, na.rm=T)) |
|  |  |
|  | # Plotting |
|  | if(!horiz) { |
|  | plot( x , dt( (x - mu[1]) / sigma[1] , df=nu[1] ) / sigma[1] , |
|  | ylim=c(0,max\_y) , cex.lab=1.5, xaxs=axs, yaxs=axs, |
|  | type="l" , col="skyblue" , lwd=1 , xlab=data\_name , ylab="Probability", |
|  | main=main, ...) |
|  | for ( i in 2:length(mu) ) { |
|  | lines(x, dt( (x - mu[i]) / sigma[i] , df=nu[i] ) / sigma[i] , |
|  | type="l" , col="skyblue" , lwd=1 ) |
|  | } |
|  | points( hist\_x , hist\_y , type="h" , lwd=3 , col="red" ,lend=1 ) |
|  | if(plot\_n) { |
|  | text( max(x) , max\_y , bquote(N==.(length(data))) , adj=c(1.1,1.1) ) |
|  | } |
|  | } else { |
|  | plot( y=x , x=dt( (x - mu[1]) / sigma[1] , df=nu[1] ) / sigma[1] , |
|  | xlim=c(0,max\_y) , cex.lab=1.5, xaxs=axs, yaxs=axs, |
|  | type="l" , col="skyblue" , lwd=1 , ylab=data\_name , xlab="Probability", |
|  | main=main, ...) |
|  | for ( i in 2:length(mu) ) { |
|  | lines(y = x, x=dt( (x - mu[i]) / sigma[i] , df=nu[i] ) / sigma[i] , |
|  | type="l" , col="skyblue" , lwd=1 ) |
|  | } |
|  | segments(x0= rep(0, length(hist\_y)), y0= hist\_x, x1 = hist\_y, y1=hist\_x, lwd=3 , col="red" ,lend=1 ) |
|  | if(plot\_n) { |
|  | text( y=max(x) , x=max\_y , bquote(N==.(length(data))) , adj=c(1.1,1.1) ) |
|  | } |
|  |  |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | # Reoder the columns of a mcmc.list coda object. |
|  | reorder\_coda <- function(s, param\_order) { |
|  | s <- lapply(s, function(chain) { |
|  | chain[, order(match(gsub("\\[.+\\]$", "", colnames(chain)), param\_order))] |
|  | }) |
|  | mcmc.list(s) |
|  | } |
|  |  |
|  | # Formats a number and returns d significant digits, keeps all |
|  | # integer digits. This is probably possible to write in a |
|  | # much more elegant manner. |
|  | sign\_digits <- function(x,d){ |
|  | s <- format(x,digits=d) |
|  | if(grepl("\\.", s) && ! grepl("e", s)) { |
|  | n\_sign\_digits <- nchar(s) - |
|  | max( grepl("\\.", s), attr(regexpr("(^[-0.]\*)", s), "match.length") ) |
|  | n\_zeros <- max(0, d - n\_sign\_digits) |
|  | s <- paste(s, paste(rep("0", n\_zeros), collapse=""), sep="") |
|  | } else if(nchar(s) < d && ! grepl("e", s)) { |
|  | s <- paste(s, ".", paste(rep("0", d - nchar(s)), collapse=""), sep="") |
|  | } |
|  | s |
|  | } |
|  |  |
|  | # rounds to d decimal places or to d significan digets depending on which leads to |
|  | # the least absolute error. |
|  | round\_or\_signif <- function(x, d) { |
|  | x\_round <- round(x, d) |
|  | x\_signif <- signif(x, d) |
|  | least\_error <- sapply(seq\_len(length(x)), function(i) { |
|  | least\_error\_i <- which.min(c(abs(x[i] - x\_signif[i]), abs(x[i] - x\_round[i]))) |
|  | # Guard against NAs and other strangeness in the the input. |
|  | if(! is.numeric(least\_error\_i) || length(least\_error\_i) != 1) { |
|  | least\_error\_i <- 1 |
|  | } |
|  | least\_error\_i |
|  | }) |
|  | x[least\_error == 1] <- x\_signif[least\_error == 1] |
|  | x[least\_error == 2] <- x\_round[least\_error == 2] |
|  | x |
|  | } |
|  |  |
|  | # Takes coda samples generates a matrix with different statistics for the |
|  | # parameters. Samples can both be a mcmc.list and a matrix with one column |
|  | # per parameter |
|  | mcmc\_stats <- function(samples, cred\_mass = 0.95, comp\_val = 0) { |
|  | samples\_mat <- as.matrix(samples) |
|  | stats <- data.frame(mean = colMeans(samples\_mat)) |
|  | stats$sd <- apply(samples\_mat, 2, sd) |
|  | cred\_mass <- rep(cred\_mass, length.out = ncol(samples\_mat)) |
|  | comp\_val <- rep(comp\_val, length.out = ncol(samples\_mat)) |
|  | stats$"HDI%" <- cred\_mass \* 100 |
|  | stats$comp <- comp\_val |
|  | stats$HDIlo <- NA |
|  | stats$HDIup <- NA |
|  | for(i in 1:ncol(samples\_mat)){ |
|  | hdi\_lim <- HDIofMCMC(samples\_mat[,i], credMass=cred\_mass[i]) |
|  | stats$HDIlo[i] <- hdi\_lim[1] |
|  | stats$HDIup[i] <- hdi\_lim[2] |
|  | stats$"%>comp"[i] <- mean(c(samples\_mat[,i] > comp\_val[i], 0, 1)) |
|  | stats$"%<comp"[i] <- mean(c(samples\_mat[,i] < comp\_val[i], 0, 1)) |
|  | } |
|  | stats$"q2.5%" <- apply(samples\_mat, 2, quantile, probs= 0.025) |
|  | stats$"q25%" <- apply(samples\_mat, 2, quantile, probs= 0.25) |
|  | stats$median <- apply(samples\_mat, 2, median) |
|  | stats$"q75%" <- apply(samples\_mat, 2, quantile, probs= 0.75) |
|  | stats$"q97.5%" <- apply(samples\_mat, 2, quantile, probs= 0.975) |
|  | stats$mcmc\_se <- NA |
|  | stats$Rhat <- NA |
|  | stats$n\_eff <- NA |
|  | if(is.mcmc.list(samples)) { |
|  | stats$mcmc\_se <- summary(samples)$statistics[,"Time-series SE"] |
|  | stats$Rhat <- gelman.diag(samples, multivariate = FALSE)$psrf[, 1] |
|  | stats$n\_eff <- as.integer(effectiveSize(samples)) |
|  | } |
|  | as.matrix(stats) # 'cause it's easier to index |
|  | } |
|  |  |
|  | # converts x to a char but returns, say, "<0.001" if x would be below low = 0.001 |
|  | num\_to\_char\_with\_lim <- function(x, low, high, digits) { |
|  | ifelse(x > high, paste(">", round(high, digits) , sep=""), |
|  | ifelse(x < low, paste("<", round(low, digits), sep=""), |
|  | as.character(round(x, digits)))) |
|  | } |
|  |  |
|  | # Takes a matrix like the one generated by the function mcmc\_stats and generates a |
|  | # version where the number have been formated into strings for pretty printing. |
|  | format\_stats <- function(s) { |
|  | s\_char <- apply(s, c(1,2), function(x) { sign\_digits(x, 2) }) |
|  | s\_char[, "comp"] <- round(s[, "comp"], 3) |
|  |  |
|  | s\_char[, "%>comp"] <- num\_to\_char\_with\_lim(s[, "%>comp"], 0.001, 0.999, 3) |
|  | s\_char[, "%<comp"] <- num\_to\_char\_with\_lim(s[, "%<comp"], 0.001, 0.999, 3) |
|  |  |
|  | s\_char |
|  | } |
|  |  |
|  | # Kruschke |
|  | HDIofICDF = function( ICDFname , credMass=0.95 , tol=1e-8 , ... ) { |
|  | # Arguments: |
|  | # ICDFname is R's name for the inverse cumulative density function |
|  | # of the distribution. |
|  | # credMass is the desired mass of the HDI region. |
|  | # tol is passed to R's optimize function. |
|  | # Return value: |
|  | # Highest density iterval (HDI) limits in a vector. |
|  | # Example of use: For determining HDI of a beta(30,12) distribution, type |
|  | # HDIofICDF( qbeta , shape1 = 30 , shape2 = 12 ) |
|  | # Notice that the parameters of the ICDFname must be explicitly named; |
|  | # e.g., HDIofICDF( qbeta , 30 , 12 ) does not work. |
|  | # Adapted and corrected from Greg Snow's TeachingDemos package. |
|  | incredMass = 1.0 - credMass |
|  | intervalWidth = function( lowTailPr , ICDFname , credMass , ... ) { |
|  | ICDFname( credMass + lowTailPr , ... ) - ICDFname( lowTailPr , ... ) |
|  | } |
|  | optInfo = optimize( intervalWidth , c( 0 , incredMass ) , ICDFname=ICDFname , |
|  | credMass=credMass , tol=tol , ... ) |
|  | HDIlowTailPr = optInfo$minimum |
|  | return( c( ICDFname( HDIlowTailPr , ... ) , |
|  | ICDFname( credMass + HDIlowTailPr , ... ) ) ) |
|  | } # Kruschke, J. K. (2011). Doing Bayesian data analysis: A |
|  | # Tutorial with R and BUGS. Elsevier Science/Academic Press. |
|  |  |
|  |  |
|  | # Kruschke |
|  | HDIofMCMC = function( sampleVec , credMass=0.95 ) { |
|  | # Computes highest density interval from a sample of representative values, |
|  | # estimated as shortest credible interval. |
|  | # Arguments: |
|  | # sampleVec |
|  | # is a vector of representative values from a probability distribution. |
|  | # credMass |
|  | # is a scalar between 0 and 1, indicating the mass within the credible |
|  | # interval that is to be estimated. |
|  | # Value: |
|  | # HDIlim is a vector containing the limits of the HDI |
|  | sortedPts = sort( sampleVec ) |
|  | ciIdxInc = floor( credMass \* length( sortedPts ) ) |
|  | nCIs = length( sortedPts ) - ciIdxInc |
|  | ciWidth = rep( 0 , nCIs ) |
|  | for ( i in 1:nCIs ) { |
|  | ciWidth[ i ] = sortedPts[ i + ciIdxInc ] - sortedPts[ i ] |
|  | } |
|  | HDImin = sortedPts[ which.min( ciWidth ) ] |
|  | HDImax = sortedPts[ which.min( ciWidth ) + ciIdxInc ] |
|  | HDIlim = c( HDImin , HDImax ) |
|  | return( HDIlim ) |
|  | } |
|  |  |
|  | # Author John Kruschke, slightly modified |
|  | plotPost = function( param\_sample\_vec , cred\_mass=0.95 , comp\_val=NULL , |
|  | HDI\_text\_place=0.7 , ROPE=NULL , yaxt=NULL , ylab=NULL , |
|  | xlab=NULL , cex.lab=NULL , cex=NULL , xlim=NULL , main=NULL , |
|  | col=NULL , border=NULL , show\_mode=FALSE , show\_median = FALSE, |
|  | show\_curve=FALSE , breaks=NULL , show\_labels = TRUE, log\_base = NULL,... ) { |
|  | # Override defaults of hist function, if not specified by user: |
|  | # (additional arguments "..." are passed to the hist function) |
|  | if ( is.null(xlab) ) xlab="Parameter" |
|  | if ( is.null(cex.lab) ) cex.lab=1.5 |
|  | if ( is.null(cex) ) cex=1.4 |
|  | if ( is.null(xlim) ) xlim=range( c( comp\_val , param\_sample\_vec ) ) |
|  | if ( is.null(main) ) main="" |
|  | if ( is.null(yaxt) ) yaxt="n" |
|  | if ( is.null(ylab) ) ylab="" |
|  | if ( is.null(col) ) col="skyblue" |
|  | if ( is.null(border) ) border="#CCF0FF" |
|  |  |
|  | postSummary = matrix( NA , nrow=1 , ncol=11 , |
|  | dimnames=list( c( xlab ) , |
|  | c("mean","median","mode", |
|  | "hdiMass","hdiLow","hdiHigh", |
|  | "comp\_val","pcGTcomp\_val", |
|  | "ROPElow","ROPEhigh","pcInROPE"))) |
|  | postSummary[,"mean"] = mean(param\_sample\_vec) |
|  | postSummary[,"median"] = median(param\_sample\_vec) |
|  | mcmcDensity = density(param\_sample\_vec) |
|  | postSummary[,"mode"] = mcmcDensity$x[which.max(mcmcDensity$y)] |
|  | HDI = HDIofMCMC( param\_sample\_vec , cred\_mass ) |
|  | if(! is.null(log\_base)) { |
|  | HDI = log\_base^HDIofMCMC( log(param\_sample\_vec, log\_base) , cred\_mass ) |
|  | } else { |
|  | HDI = HDIofMCMC( param\_sample\_vec , cred\_mass ) |
|  | } |
|  |  |
|  | postSummary[,"hdiMass"]=cred\_mass |
|  | postSummary[,"hdiLow"]=HDI[1] |
|  | postSummary[,"hdiHigh"]=HDI[2] |
|  |  |
|  | # Plot histogram. |
|  | if ( is.null(breaks) ) { |
|  | if(! is.null(log\_base) ) { |
|  | log\_param\_sample\_vec <- log(param\_sample\_vec, log\_base) |
|  | HDI95 = HDIofMCMC( log\_param\_sample\_vec , 0.95 ) |
|  | breaks = c( seq( from=min(log\_param\_sample\_vec) , to=max(log\_param\_sample\_vec) , |
|  | by=(HDI95[2]-HDI95[1])/18 ) , max(log\_param\_sample\_vec) ) |
|  | } else { |
|  | HDI95 = HDIofMCMC( param\_sample\_vec , 0.95 ) |
|  | breaks = c( seq( from=min(param\_sample\_vec) , to=max(param\_sample\_vec) , |
|  | by=(HDI95[2]-HDI95[1])/18 ) , max(param\_sample\_vec) ) |
|  | } |
|  |  |
|  | } |
|  | if ( !show\_curve ) { |
|  | par(xpd=NA) |
|  | if(! is.null(log\_base)) { |
|  | old\_par <- par(lab=c(6, 5, 7)) |
|  | histinfo <- hist( log(param\_sample\_vec, log\_base) , xaxt = "n",xlab=xlab , yaxt=yaxt , ylab=ylab , |
|  | freq=F , border=border , col=col , xlim=log(xlim, log\_base) , main=main , cex=cex , cex.lab=cex.lab , |
|  | breaks=breaks , ... ) |
|  | log\_labels = as.character(fractions(log\_base^axTicks(1, ), max.denominator = 2^64)) |
|  | extreme\_axis\_ticks <- log\_base^axTicks(1) > 9999 | log\_base^axTicks(1) < 1/9999 |
|  | log\_labels[extreme\_axis\_ticks] <- paste(log\_base, "^", axTicks(1)[extreme\_axis\_ticks], sep="") |
|  | axis(1, at = axTicks(1), labels = log\_labels) |
|  | par(old\_par) |
|  | } else { |
|  | histinfo = hist( param\_sample\_vec , xlab=xlab , yaxt=yaxt , ylab=ylab , |
|  | freq=F , border=border , col=col , |
|  | xlim=xlim , main=main , cex=cex , cex.lab=cex.lab , |
|  | breaks=breaks , ... ) |
|  | } |
|  | } |
|  | if ( show\_curve ) { |
|  | par(xpd=NA) |
|  | histinfo = hist( param\_sample\_vec , plot=F ) |
|  | densCurve = density( param\_sample\_vec , adjust=2 ) |
|  | plot( densCurve$x , densCurve$y , type="l" , lwd=5 , col=col , bty="n" , |
|  | xlim=xlim , xlab=xlab , yaxt=yaxt , ylab=ylab , |
|  | main=main , cex=cex , cex.lab=cex.lab , ... ) |
|  | } |
|  | cenTendHt = 0.9\*max(histinfo$density) |
|  | cvHt = 0.7\*max(histinfo$density) |
|  | ROPEtextHt = 0.55\*max(histinfo$density) |
|  | # Display mean or mode: |
|  | if ( show\_mode==T ) { |
|  | dres = density( param\_sample\_vec ) |
|  | modeParam = dres$x[which.max(dres$y)] |
|  | if(show\_labels) { |
|  | text\_label <- bquote(mode==.(sign\_digits(modeParam,2))) |
|  | } else { |
|  | text\_label <- sign\_digits(modeParam,2) |
|  | } |
|  | if(! is.null(log\_base)) { |
|  | text( log(modeParam, log\_base) , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } else { |
|  | text( modeParam , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } |
|  | } else if(show\_median) { |
|  | medianParam = median( param\_sample\_vec ) |
|  | if(show\_labels) { |
|  | text\_label <- bquote(median==.(sign\_digits(medianParam,2))) |
|  | } else { |
|  | text\_label <- sign\_digits(medianParam,2) |
|  | } |
|  | if(! is.null(log\_base)) { |
|  | text( log(medianParam, log\_base) , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } else { |
|  | text( medianParam , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } |
|  | } else { # Show the mean |
|  | meanParam = mean( param\_sample\_vec ) |
|  | if(show\_labels) { |
|  | text\_label <- bquote(mean==.(sign\_digits(meanParam,2))) |
|  | } else { |
|  | text\_label <- sign\_digits(meanParam,2) |
|  | } |
|  | if(! is.null(log\_base)) { |
|  | text( log(meanParam, log\_base) , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } else { |
|  | text( meanParam , cenTendHt , text\_label, adj=c(.5,0) , cex=cex ) |
|  | } |
|  | } |
|  |  |
|  |  |
|  | # Display the comparison value. |
|  | if ( !is.null( comp\_val ) ) { |
|  | cvCol = "darkgreen" |
|  | pcgtcomp\_val = round( 100 \* sum( param\_sample\_vec > comp\_val ) |
|  | / length( param\_sample\_vec ) , 1 ) |
|  | pcltcomp\_val = 100 - pcgtcomp\_val |
|  | if(! is.null(log\_base)) { |
|  | comp\_val\_pos <- log(comp\_val, log\_base) |
|  | } else { |
|  | comp\_val\_pos <- comp\_val |
|  | } |
|  |  |
|  | lines( c(comp\_val\_pos,comp\_val\_pos) , c(0.96\*cvHt,0) , |
|  | lty="dotted" , lwd=1 , col=cvCol ) |
|  | text( comp\_val\_pos , cvHt , |
|  | bquote( .(pcltcomp\_val)\*"% < " \* |
|  | .(signif(comp\_val,2)) \* " < "\*.(pcgtcomp\_val)\*"%" ) , |
|  | adj=c(pcltcomp\_val/100,0) , cex=0.8\*cex , col=cvCol ) |
|  | postSummary[,"comp\_val"] = comp\_val |
|  | postSummary[,"pcGTcomp\_val"] = ( sum( param\_sample\_vec > comp\_val ) |
|  | / length( param\_sample\_vec ) ) |
|  | } |
|  | # Display the ROPE. |
|  | if ( !is.null( ROPE ) ) { |
|  | ropeCol = "darkred" |
|  | pcInROPE = ( sum( param\_sample\_vec > ROPE[1] & param\_sample\_vec < ROPE[2] ) |
|  | / length( param\_sample\_vec ) ) |
|  | if(! is.null(log\_base)) { |
|  | ROPE\_pos <- log(ROPE, log\_base) |
|  | } else { |
|  | ROPE\_pos <- ROPE |
|  | } |
|  |  |
|  | lines( c(ROPE\_pos[1],ROPE\_pos[1]) , c(0.96\*ROPEtextHt,0) , lty="dotted" , lwd=2 , |
|  | col=ropeCol ) |
|  | lines( c(ROPE\_pos[2],ROPE\_pos[2]) , c(0.96\*ROPEtextHt,0) , lty="dotted" , lwd=2 , |
|  | col=ropeCol) |
|  | text( mean(ROPE\_pos) , ROPEtextHt , |
|  | bquote( .(round(100\*pcInROPE))\*"% in ROPE" ) , |
|  | adj=c(.5,0) , cex=1 , col=ropeCol ) |
|  |  |
|  | postSummary[,"ROPElow"]=ROPE[1] |
|  | postSummary[,"ROPEhigh"]=ROPE[2] |
|  | postSummary[,"pcInROPE"]=pcInROPE |
|  | } |
|  | # Display the HDI. |
|  | if(! is.null(log\_base)) { |
|  | HDI\_pos <- log(HDI, log\_base) |
|  | } else { |
|  | HDI\_pos <- HDI |
|  | } |
|  |  |
|  | lines( HDI\_pos , c(0,0) , lwd=4 ) |
|  | if(show\_labels) { |
|  | text( mean(HDI\_pos) , 0 , bquote(.(100\*cred\_mass) \* "% HDI" ) , |
|  | adj=c(.5,-1.7) , cex=cex ) |
|  | } |
|  | text( HDI\_pos[1] , 0 , bquote(.(sign\_digits(HDI[1],2))) , |
|  | adj=c(HDI\_text\_place,-0.5) , cex=cex ) |
|  | text( HDI\_pos[2] , 0 , bquote(.(sign\_digits(HDI[2],2))) , |
|  | adj=c(1.0-HDI\_text\_place,-0.5) , cex=cex ) |
|  | par(xpd=F) |
|  | # |
|  | #return( postSummary ) |
|  | return(invisible()) |
|  | } |

It’s fiendishly easy to set-up and run. We’ll go back to our original dat

object and divide it up by column. Column 1 those who got COVID by condition “Placebo” and “Vaccine” and column 2 those who didn’t. To keep this from being **tl;dr** I’m not going to go into too much detail on jags and runjags.

If you need a good tutorial consult a reference like DBDA. I’ll only highlight

the key points. For purposes of this post I’m going to use a flat, uninformed prior in all cases (this time the specific line is

theta[i] ~ dbeta(1, 1)). One of the nice aspects of bayesian inference is you can express your prior thinking/knowledge mathematically, then

let the data inform your thinking. Given the vaccine has already been involved in significant phase I and II trials it wouldn’t be unusual

to have a prior that expressed at least a little confidence that it had some effect. But we’ll pretend we know nothing and that any outcome is equally likely. We’re back to flipping coins again.

We’re going to explore the probabilities that the percentage of positive cases among those who received the vaccine is the same as the percentage of positive cases among those who received the vaccine. The model is constructed so that we could conceivably have more than two possibilities (e.g. easy to extend

to a case where we had two different vaccines + a placebo or two different doses of the same vaccine plus placebo). That is very likely to be modeled later or when more data are in. We’ll “monitor” results for the infection rate placebo and vaccine (theta[1] and theta[2] respectively).

As well as the raw predictions of how many people (x\_pred[1] and x\_pred[2]), not strictly necessary but fun to watch.

This is a simple model with just four data elements we need to enter but we’ll still run it in 4 chains spread across 4 cores. If you’re following along I’m suppressing some messages about not choosing different random seeds per chain (run\_jags will pick some), and that we used the same initial value for all chains (also not a worry). You should also run plot(my\_results) to check the diagnostics for convergence and auto-correlation. I did, they’re fine but to save screen real estate I won’t include them.

# Setting up the data dat

## COVID No COVID

## Placebo 34 14966

## Vaccine 19 14981

got\_covid <- dat[,1]

not\_covid <- dat[,2] got\_covid

## Placebo Vaccine ## 34 19

not\_covid

## Placebo Vaccine ## 14966 14981

# The model string written in the JAGS language model\_string <- "model {

for(i in 1:length(got\_covid)) {

got\_covid[i] ~ dbinom(theta[i], not\_covid[i]) theta[i] ~ dbeta(1, 1)

x\_pred[i] ~ dbinom(theta[i], not\_covid[i])

}

}"

my\_results <- runjags::run.jags(model\_string,

sample = 10000, n.chains=4, method="parallel",

monitor = c("theta", "x\_pred"), data = list(got\_covid = got\_covid,

not\_covid = not\_covid))

|  |  |  |
| --- | --- | --- |
| ## Calling 4 simulations using the parallel method... |  | |
| ## Following the progress of chain 1 (the program will | wait for | all |
| chains |  |  |
| ## to finish before continuing): |  |  |
| ## Welcome to JAGS 4.3.0 on Wed Nov 4 15:34:07 2020 |  |  |
| ## JAGS is free software and comes with ABSOLUTELY NO | WARRANTY |  |
| ## Loading module: basemod: ok |  |  |
| ## Loading module: bugs: ok |  |  |
| ## . . Reading data file data.txt |  |  |
| ## . Compiling model graph |  |  |
| ## Resolving undeclared variables |  |  |
| ## Allocating nodes |  |  |
| ## Graph information: |  |  |
| ## Observed stochastic nodes: 2 |  |  |
| ## Unobserved stochastic nodes: 4 |  |  |
| ## Total graph size: 9 |  |  |
| ## . Reading parameter file inits1.txt |  |  |
| ## . Initializing model |  |  |
| ## . Adapting 1000 |  |  |
| ## - | | 1000 |  |
| ## ++++++++++++++++++++++++++++++++++++++++++++++++++ | 100% |  |
| ## Adaptation successful |  |  |
| ## . Updating 4000 |  |  |
| ## - | | 4000 |  |
| ## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | 100% |  |
| ## . . . Updating 10000  ## - | | 10000 |  |
| ## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | 100% |  |
| ## . . . . Updating 0 |  |  |

## . Deleting model ## .

## All chains have finished

## Simulation complete. Reading coda files... ## Coda files loaded successfully

## Calculating summary statistics...

## Calculating the Gelman-Rubin statistic for 4 variables....

## Finished running the simulation my\_results

##

## JAGS model summary statistics from 40000 samples (chains = 4;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| adapt+burnin  ## | = 5000): |  | | | | |
| ## | Lower95 | Median | Upper95 | Mean | SD | Mode |
| MCerr |  |  |  |  |  |  |
| ## theta[1] | 0.0015847 | 0.0023125 | 0.0031218 | 0.0023357 | 0.00039503 | -- |
| 2.5904e-06 |  |  |  |  |  |  |
| ## theta[2] | 0.00077049 | 0.0013122 | 0.001919 | 0.0013345 | 0.00029786 | -- |
| 1.9645e-06 |  |  |  |  |  |  |
| ## x\_pred[1] | 18 | 34 | 50 | 34.949 | 8.3756 | 33 |
| 0.048986 |  |  |  |  |  |  |
| ## x\_pred[2] | 8 | 20 | 32 | 20.036 | 6.3042 | 19 |
| 0.037185 |  |  |  |  |  |  |
| ## |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | MC%ofSD | SSeff | AC.10 | psrf |
| ## | theta[1] | 0.7 | 23255 | -0.0023991 | 1.0001 |
| ## | theta[2] | 0.7 | 22988 | 0.0022903 | 1.0001 |
| ## | x\_pred[1] | 0.6 | 29234 | -0.0046137 | 1.0001 |
| ## | x\_pred[2] | 0.6 | 28743 | 0.0047744 | 1.0001 |
| ## |  |  |  |  |  |

## Total time taken: 1.3 seconds

# plot(my\_results) ## diagnostics were checked.

Okay that all looks quite complex, what do we do now? Let’s first investigate the things we care most about. Remember theta[1] & theta[2] represent our infection rates theta[1] = 0.0023357 is our rate among

those who got the placebo and theta[2] = 0.0013345 is our

rate among those who got the vaccine. Those are the mean values and the medians are similar. The x\_pred values are estimates of the case counts.

We can use tidybayes::tidy\_draws to extract the results of our 40,000 chains and pipe it through select to get the columns we want with the names we’d

like. As much as I like Greek \(\theta\) gets old after awhile. At the same time we can compute via a mutate statement what we really want to know which is the % difference in infection rates which we’ll put in a column called diff\_rate.

Now when we pass this cleaned up data to bayestestR::describe\_ posterior(results1)

we get back a table that is a little easier to read. Focus on just the line for diff\_rate.

results1 <- tidybayes::tidy\_draws(my\_results) %>% select(placebo\_rate = `theta[1]`,

vaccine\_rate = `theta[2]`, placebo\_cases = `x\_pred[1]`, vaccine\_cases = `x\_pred[2]`) %>%

mutate(diff\_rate = (placebo\_rate - vaccine\_rate) / placebo\_rate \* 100)

glimpse(results1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## | Rows: 40,000 |  | | | |
| ## | Columns: 5 |
| ## | $ placebo\_rate | 0.00319585, | 0.00300531, | 0.00293815, | 0.00228591, |
| 0.00238… | |  |  |  |  |
| ## $ vaccine\_rate | | 0.00108651, | 0.00131015, | 0.00134464, | 0.00137949, |
| 0.00112… | |  |  |  |  |
| ## $ placebo\_cases | | 52, 47, 41, | 36, 28, 21, | 37, 34, 31, | 34, 48, 37, 40, |
| 33,… | |  |  |  |  |
| ## $ vaccine\_cases | | 9, 20, 23, 22, 11, 28, 21, 13, 20, 22, 27, 24, 15, | | | |
| 30, … | |  | | | |
| ## $ diff\_rate | | 66.002472, 56.405496, 54.235148, 39.652480, | | | |
| 52.911123, … | |  | | | |

bayestestR::describe\_posterior(results1, ci = 0.95) ## # Description of Posterior Distributions

##

## Parameter | Median | 95% CI | pd | 89%

ROPE | % in ROPE

## - -

-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## placebo\_rate | | 0.002 | | | [ 0.002, | 0.003] | | | 100.00% | | | [-0.100, |
| 0.100] | 100 |  |  |  |  |  |  |  |  |
| ## vaccine\_rate | | 0.001 | | | [ 0.001, | 0.002] | | | 100.00% | | | [-0.100, |
| 0.100] | 100 |  |  |  |  |  |  |  |  |
| ## placebo\_cases | | 34.000 | | | [20.000, | 52.000] | | | 100.00% | | | [-0.100, |
| 0.100] | 0 |  |  |  |  |  |  |  |  |
| ## vaccine\_cases | | 20.000 | | | [10.000, | 34.000] | | | 100.00% | | | [-0.100, |
| 0.100] | 0 |  |  |  |  |  |  |  |  |
| ## diff\_rate | | 43.316 | | | [ 7.715, | 71.077] | | | 97.97% | | | [-0.100, |
| 0.100] | 0 |  |  |  |  |  |  |  |  |

Given our data and 40,000 samples and assuming we had zero prior knowledge or estimation of effectiveness, then we have a median estimate of the % difference

in infection rates = 43.316. That’s about what

we would expect given our earlier investigation. The columns we **really** want

to use are the “95% CI” and “pd” columns for diff\_rate. CI in a bayesian framework is credible interval not confidence interval. Since 95% of our chains wind up in

that interval we can say given our data there’s a 95% probability that diff\_rate

lies in its range. No, they are not the same thing as a confidence interval. The Probability of Direction (pd) is an index of effect existence, ranging from 50% to 100%, representing the certainty with which an effect goes in a particular direction (i.e., is positive or negative). We can be very confident that the vaccine does have a positive effect.

Plotting the distribution or range of diff\_rate may also help the reader “see” the results. With the data we have (and remember I have been using a 19/34 split) you can see that while there’s a chance that the vaccine has no effect the evidence (the data) supports the notion that it does.

plot(bayestestR::hdi(results1$diff\_rate,

ci = c(.89, .95, .99)

)

)

