Package: truelies (via r-universe)

August 15, 2026

Type Package

Title Bayesian Methods to Estimate the Proportion of Liars in Coin Flip Experiments

Version 0.2.1

Author David Hugh-Jones <davidhughjones@gmail.com>

Maintainer David Hugh-Jones <davidhughjones@gmail.com>

Description Implements Bayesian methods, described in Hugh-Jones (2019) <doi:10.1007/s40881-019-00069-x>, for estimating the proportion of liars in coin flip-style experiments, where subjects report a random outcome and are paid for reporting a ``good'' outcome.

License MIT + file LICENSE

URL https://github.com/hughjonesd/truelies

BugReports https://github.com/hughjonesd/truelies/issues

Imports hdrcde

Suggests dplyr, ggplot2, MASS, purrr, tidyr

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Repository https://r-multiverse-staging.r-universe.dev

Date/Publication 2026-05-26 15:10:25 UTC

RemoteUrl https://github.com/hughjonesd/truelies

RemoteRef 4a0b41cde2d32c28662645c1b8780192e57400a2

RemoteSha 4a0b41cde2d32c28662645c1b8780192e57400a2

Contents

`compare_dists` *Calculate probability that one posterior is larger than another*

Description

Given two distributions with density functions ϕ_1, ϕ_2, this calculates:

$$\int_0^1 \int_0^{l_1} \phi_1(l_1)\phi_2(l_2) dl_2 dl_1,$$

the probability that the value of the first distribution is greater.

Usage

```
compare_dists(dist1, dist2)
```

Arguments

`dist1`	Density of distribution 1, as a one-argument function.
`dist2`	Density of distribution 2.

Value

A probability scalar.

Examples

```
d1 <- update_prior(30, 50, P = 0.5, prior = stats::dunif)
d2 <- update_prior(25, 40, P = 0.5, prior = stats::dunif)
compare_dists(d1, d2)
```

difference_dist *Find density of the difference of two distributions*

Description

Given two probability density functions `dist1` and `dist2`, `difference_dist` returns the density of "dist1 - dist2".

Usage

```
difference_dist(dist1, dist2)
```

Arguments

`dist1`, `dist2`	Probability density functions

Details

At the moment this only works when dist1 and dist2 are defined on `[0, 1]`.

Value

A probability density function defined on `[-1, 1]`.

Examples

```
d1 <- update_prior(30, 50, P = 0.5, prior = stats::dunif)
d2 <- update_prior(32, 40, P = 0.5, prior = stats::dunif)
dd <- difference_dist(d1, d2)
dist_hdr(dd, 0.95)
```

dist_hdr *Compute highest density region for a density function*

Description

This is a wrapper for `hdrcde::hdr`. The highest density region is the interval that covers `conf_level` of the data and has the highest average density. See:

Usage

```
dist_hdr(dist, conf_level, bounds = attr(dist, "limits"))
```

Arguments

dist	A one-argument function
conf_level	A scalar between 0 and 1
bounds	A length 2 vector of the bounds of the distribution's support

Details

Rob J Hyndman (1996) "Computing and graphing highest density regions". American Statistician, 50, 120-126.

Value

A length 2 vector of region endpoints

Examples

```
d1 <- update_prior(33, 50, P = 0.5, prior = stats::dunif)
dist_hdr(d1, 0.95)
```

---

dist_mean *Find mean of a probability density function*

Description

Find mean of a probability density function

Usage

```
dist_mean(dist, l = attr(dist, "limits")[1], r = attr(dist, "limits")[2])
```

Arguments

dist	A one-argument function returned from update_prior()
l	Lower bound of the density's support
r	Upper bound of the density's support

Value

A scalar

Examples

```
d1 <- update_prior(10, 40, P = 5/6, prior = stats::dunif)
dist_mean(d1)
```

dist_quantile *Find quantiles given a probability density function*

Description

Find quantiles given a probability density function

Usage

```
dist_quantile(dist, probs, bounds = attr(dist, "limits"))
```

Arguments

dist	A one argument function
probs	A vector of probabilities
bounds	A length 2 vector of the bounds of the distribution's support

Value

A vector of quantiles

Examples

```
d1 <- update_prior(33, 50, P = 0.5, prior = stats::dunif)
dist_quantile(d1, c(0.025, 0.975))
```

empirical_bayes *Estimate proportions of liars in multiple samples using empirical Bayes*

Description

This function creates a prior by fitting a Beta distribution to the heads/N vector, using MASS::fitdistr(). The prior is then updated using data from each individual sample to give the posterior distributions.

Usage

```
empirical_bayes(heads, ...)

## Default S3 method:
empirical_bayes(heads, N, P, ...)

## S3 method for class 'formula'
empirical_bayes(formula, data, P, subset, ...)
```

Arguments

heads	A vector of numbers of the good outcome reported
...	Ignored
N	A vector of sample sizes
P	Probability of *bad* outcome
formula	A two-sided formula of the form heads ~ group. heads is a logical vector specifying whether the "good" outcome was reported. group specifies the sample.
data	A data frame or matrix. Each row represents one individual.
subset	A logical or numeric vector specifying the subset of data to use

Details

The formula interface allows calling the function directly on experimental data.

Value

A list with two components:

- prior, the calculated empirical prior (of class densityFunction).
- posterior, a list of posterior distributions (objects of class densityFunction). If heads was named, the list will have the same names.

Examples

```
heads <- c(Baseline = 30, Treatment1 = 38, Treatment2 = 45)
N <- c(50, 52, 57)
res <- empirical_bayes(heads, N, P = 0.5)

compare_dists(res$posteriors$Baseline, res$posteriors$Treatment1)
plot(res$prior, ylim = c(0, 4), col = "grey", lty = 2)
plot(res$posteriors$Baseline, add = TRUE, col = "blue")
plot(res$posteriors$Treatment1, add = TRUE, col = "orange")
plot(res$posteriors$Treatment2, add = TRUE, col = "red")


# starting from raw data:
raw_data <- data.frame(
        report = sample(c("heads", "tails"),
          size = 300,
          replace = TRUE,
          prob = c(.8, .2)
        ),
        group = rep(LETTERS[1:10], each = 30)
    )
empirical_bayes(I(report == "heads") ~ group, data = raw_data, P = 0.5)
```

power_calc *Calculate power to detect non-zero lying*

Description

This uses simulations to estimate the power to detect a given level of lying in a sample of size `N` by this package's methods.

Usage

```
power_calc(N, P, lambda, alpha = 0.05, prior = stats::dunif, nsims = 200)
```

Arguments

`N`	Total number in sample
`P`	Probability of *bad* outcome
`lambda`	Probability of a subject lying
`alpha`	Significance level to use for the null hypothesis
`prior`	Prior over lambda. A function which takes a vector of values between 0 and 1, and returns the probability density. The default is the uniform distribution.
`nsims`	Number of simulations to run

Value

Estimated power, a scalar between 0 and 1.

Examples

```
power_calc(N = 50, P = 0.5, lambda = 0.2)
```

power_calc_difference *Estimate power to detect differences in lying between two samples*

Description

Using simulations, estimate power to detect differences in lying using `compare_dists()`, given values for λ, the probability of lying, in each sample.

Usage

```
power_calc_difference(
  N1,
  N2 = N1,
  P,
  lambda1,
  lambda2,
  alpha = 0.05,
  alternative = c("two.sided", "greater", "less"),
  prior = stats::dunif,
  nsims = 200
)
```

Arguments

N1	N of sample 1
N2	N of sample 2
P	Probability of *bad* outcome
lambda1	Probability of lying in sample 1
lambda2	Probability of lying in sample 2
alpha	Significance level
alternative	"two.sided", "greater" (sample 1 is greater), or "less". Can be abbreviated
prior	Prior over lambda. A function which takes a vector of values between 0 and 1, and returns the probability density. The default is the uniform distribution.
nsims	Number of simulations to run

Value

Estimated power, a scalar between 0 and 1.

Examples

```
power_calc_difference(N1 = 100, P = 0.5, lambda = 0, lambda2 = 0.25)
```

---

print.densityFunction *Print/plot an object of class* densityFunction.

Description

Print/plot an object of class densityFunction.

Usage

```
## S3 method for class 'densityFunction'
print(x, ...)

## S3 method for class 'densityFunction'
plot(x, ...)
```

Arguments

x	The object
...	Unused

Examples

```
d1 <- update_prior(33, 50, P = 0.5, prior = stats::dunif)
d1
plot(d1)

# show the actual R code (techies only)
unclass(d1)
```

---

update_prior *Calculate posterior distribution of the proportion of liars*

Description

update_prior uses the equation for the posterior:

$$\phi(\lambda|R; N, P) = Pr(R|\lambda; N, P)\phi(\lambda) / \int Pr(R|\lambda'; N, P)\phi(\lambda')d\lambda'$$

where ϕ is the prior and $Pr(R|\lambda; N, P)$ is the probability of R reports of heads given that people lie with probability λ:

$$Pr(R|\lambda; N, P) = binom(N, (1 - P) + \lambda P)$$

Usage

```
update_prior(heads, N, P, prior = stats::dunif, npoints = 1000)
```

Arguments

heads	Number of good outcomes reported
N	Total number in sample
P	Probability of *bad* outcome
prior	Prior over lambda. A function which takes a vector of values between 0 and 1, and returns the probability density. The default is the uniform distribution.
npoints	How many points to integrate on?

Value

The probability density of the posterior distribution, as a one-argument function.

Examples

```
posterior <- update_prior(heads = 30, N = 50, P = 0.5, prior = stats::dunif)
plot(posterior)
```

Index