Package ‘Pinference’

November 30, 2025

Title Probability Inference for Propositional Logic

Version 0.2.6

Description Implementation of T. Hailperin's procedure to calculate lower and upper bounds of the probability for a propositional-logic expression, given equality and inequality constraints on the probabilities for other expressions. Truth-valuation is included as a special case. Applications range from decision-making and probabilistic reasoning, to pedagogical for probability and logic courses. For more details see T. Hailperin (1965) <doi:10.1080/00029890.1965.11970533>, T. Hailperin (1996) ``Sentential Probability Logic'' ISBN:0-934223-45-9, and package documentation. Requires the 'lpSolve' package.

License AGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports lpSolve

VignetteBuilder knitr

URL https://pglpm.github.io/Pinference/,
https://github.com/pglpm/Pinference/

Suggests knitr, rmarkdown

NeedsCompilation no

Author PierGianLuca Porta Mana [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-6070-0784>)

Maintainer PierGianLuca Porta Mana <pgl@portamana.org>

Repository CRAN

Date/Publication 2025-11-30 16:40:26 UTC

Contents

`inferP` *Calculate lower and upper probability bounds*

Description

`inferP()` calculates the minimum and maximum allowed values of the probability for a propositional-logic expression conditional on another one, given numerical or equality constraints for the conditional probabilities for other propositional-logic expressions.

Usage

```
inferP(target, ..., solidus = TRUE)
```

Arguments

`target`	The target probability expression (see Details).
`...`	Probability constraints (see Details).
`solidus`	logical. If `TRUE` (default), the symbol `|` is used to introduce the conditional in the probability; in this case any use of `||` for the 'or'-connective will lead to an error. If `FALSE`, the symbol `~` is used to introduce the conditional; in this case the symbols `|`, `||` can be used for the 'or'-connective.

Details

The function takes as first argument the probability for a logical expression, conditional on another expression, and as subsequent (optional) arguments the constraints on the probabilities for other logical expressions. Propositional logic is intended here.

The function uses the `lpSolve::lp()` function from the **lpSolve** package.

Logical expressions:

A propositional-logic expression is a combination of atomic propositions by means of logical connectives. Atomic propositions can have any name that satisfies R syntax for *object names*. Examples:

```
a
A
hypothesis1
coin.lands.tails
coin_lands_heads
`tomorrow it rains` # note the backticks
```

Available logical connectives are "not" (negation, "$\neg$"), "and" (conjunction, "$\wedge$"), "or" (disjunction, "$\vee$"), "if-then" (implication, "$\Rightarrow$") which internally is simply defined as "x or not-y". Two kinds of notation are available:

Arithmetic notation: should be used with argument `solidus = TRUE`:

- Not: `-`

- And: `*`
- Or: `+`
- If-then: `>`

Logical notation (see base::logical): should be used with the argument `solidus = FALSE`:

- Not: `!`
- And: `&` or `&&`
- Or: `|` or `||`
- If-then: two-argument function `ifthen()`

The two kinds of notation can actually be mixed (only `|` and `||` are not allowed if `solidus = TRUE`), but **beware** of very unusual connective precedence in that case; for example, `!a + b` is interpreted as `!(a + b)`. If you use mixed notation you should explicitly group with parentheses to ensure the desired connective precedence. A warning is issued when `!` is used for negation together with `+`, `*`, because the grouping is dangerously counter-intuitive in this case.

Examples of logical expressions:

```
a
a & b
(a + hypothesis1) & -A
red.ball & ((a > -b) + c)
```

Probabilities of logical expressions:

The probability of an expression X conditional on an expression Y in entered with syntax similar to the common mathematical notation $\mathrm{P}(X|Y)$. The solidus "`|`" is used to separate the conditional (note that in usual R syntax such symbol stands for logical "or" instead). If the argument `solidus = FALSE` is given in the function, then the tilde "`~`" is used instead of the solidus (note that in usual R syntax such symbol introduces a formula instead). For instance

$\mathrm{P}(\neg a \lor b \mid c \land H)$

can be entered in the following ways, among others (extra spaces added just for clarity):

```
P(-a + b  |  c * H)
```

or, if argument `solidus = FALSE`:

```
P(!a | b  ~  c & H)
P(!a || b  ~  c && H)
```

It is also possible to use `Pr`, `p`, `pr`, `F`, `f`, `T` (for "truth"), or `t` instead of `P`.

Probability constraints:

Each probability constraint can have one of these four forms:

```
P(X | Z) = [number between 0 and 1]

P(X | Z) = P(Y | Z)

P(X | Z) = P(Y | Z) * [positive number]

P(X | Z) = P(Y | Z) / [positive number]
```

where `X`, `Y`, `Z` are logical expressions. Note that the conditionals on the left and right sides must be the same. Inequalities <= >= are also allowed instead of equalities.

See the accompanying vignette for more interesting examples.

Value

A vector of `min` and `max` values for the target probability, or `NA` if the constraints are mutually contradictory. If `min` and `max` are `0` and `1` then the constraints do not restrict the target probability in any way.

References

T. Hailperin: *Best Possible Inequalities for the Probability of a Logical Function of Events*. Am. Math. Monthly 72(4):343, 1965 doi:10.1080/00029890.1965.11970533.

T. Hailperin: *Sentential Probability Logic: Origins, Development, Current Status, and Technical Applications*. Associated University Presses, 1996 https://archive.org/details/hailperin1996-Sentential_probability_logic/.

Examples

```
## No constraints
inferP(
  target = P(a | h)
)

## Trivial example with inequality constraint
inferP(
  target = P(a | h),
  P(-a | h) >= 0.2
)

#' ## The probability of an "and" is always less
## than the probabilities of the and-ed propositions:
inferP(
  target = P(a & b | h),
  P(a | h) == 0.3,
  P(b | h) == 0.6
)

## P(a & b | h) is completely determined
## by P(a | h) and P(b | a & h):
inferP(
    target = P(a & b | h),
    P(a | h) == 0.3,
    P(b | a & h) == 0.2
)

## Logical implication (modus ponens)
inferP(
  target = P(b | I),
  P(a | I) == 1,
  P(a > b | I) == 1
)

## Cut rule of sequent calculus
inferP(
```

```
  target = P(X + Y | I & J),
  P(A & X | I) == 1,
  P(Y | A & J) == 1
)

## Solution to the Monty Hall problem (see accompanying vignette):
inferP(
    target = P(car2  |  you1 & host3 & I),
    ##
    P(car1 & car2  |  I) == 0,
    P(car1 & car3  |  I) == 0,
    P(car2 & car3  |  I) == 0,
    P(car1 + car2 + car3  |  I) == 1,
    P(host1 & host2 | I) == 0,
    P(host1 & host3 | I) == 0,
    P(host2 & host3 | I) == 0,
    P(host1 + host2 + host3  |  I) == 1,
    P(host1  |  you1 & I) == 0,
    P(host2  |  car2 & I) == 0,
    P(host3  |  car3 & I) == 0,
    P(car1  |  I) == P(car2  |  I),
    P(car2  |  I) == P(car3  |  I),
    P(car1  |  you1 & I) == P(car2  |  you1 & I),
    P(car2  |  you1 & I) == P(car3  |  you1 & I),
    P(host2  |  you1 & car1 & I) == P(host3  |  you1 & car1 & I)
)
```

Index