Package 'bayesAB'

October 12, 2022

Type Package

Title Fast Bayesian Methods for AB Testing

Version 1.1.3

Date 2021-06-24

Description A suite of functions that allow the user to analyze A/B test data in a Bayesian framework. Intended to be a drop-in replacement for common frequentist hypothesis test such as the t-test and chi-sq test.

License MIT + file LICENSE

Imports Rcpp (>= 0.12.4), ggplot2 (>= 3.2.0), methods, rlang (>= 0.4.0)

LinkingTo Rcpp

RoxygenNote 7.1.1

Encoding UTF-8

URL https://github.com/FrankPortman/bayesAB

BugReports https://github.com/FrankPortman/bayesAB/issues

Suggests testthat, knitr, rmarkdown, magrittr, plumber (>= 0.3.0)

VignetteBuilder knitr

NeedsCompilation yes

Author Frank Portman [aut, cre]

Maintainer Frank Portman <frank1214@gmail.com>

Repository CRAN

Date/Publication 2021-06-25 00:50:02 UTC

R topics documented:

`banditize` *Create a multi-armed Bayesian bandit object.*

Description

Fit a multi-armed bandit object based on a bayesTest which can serve recommendations and adapt to new data.

Usage

```
banditize(bT, param, higher_is_better = TRUE)
```

Arguments

`bT`	a bayesTest object
`param`	which model parameter (posterior) to evaluate; defaults to first param
`higher_is_better`	is a higher value of ‘param‘ equivalent to a better choice?

Details

`banditize` is an 'object-oriented' implementation of multi-armed bandits in `bayesAB`. It is useful in conjunction with a Shiny app or Plumber deployment. The object itself is mutable and can adapt/learn from new data without having to re-assign the variable.

Comes with 5 methods:

- `serveRecipe()`: serves a recipe to show your user based on samples from both posteriors.
- `setResults(results)`: set results for one or more recipes for one or more instances of feedback. Used to update bandit.

- `getBayesTest()`: returns most updated `bayesTest` object.
- `getOriginalTest()`: returns original `bayesTest` object without any updates.
- `getUpdates()`: returns a summarized version of all updates this bandit has processed.

Value

A bayesBandit object.

Examples

```
A_binom <- rbinom(100, 1, .5)
B_binom <- rbinom(100, 1, .6)

AB1 <- bayesTest(A_binom, B_binom, priors = c('alpha' = 1, 'beta' = 1), distribution = 'bernoulli')

binomialBandit <- banditize(AB1)
binomialBandit$serveRecipe()
binomialBandit$setResults(list('A' = c(1, 0, 1, 0, 0), 'B' = c(0, 0, 0, 0, 1)))
```

`bayesAB` *bayesAB: Fast Bayesian Methods for A/B Testing*

Description

bayesAB provides a suite of functions that allow the user to analyze A/B test data in a Bayesian framework. bayesAB is intended to be a drop-in replacement for common frequentist hypothesis test such as the t-test and chi-sq test.

Details

Bayesian methods provide several benefits over frequentist methods in the context of A/B tests - namely in interpretability. Instead of p-values you get direct probabilities on whether A is better than B (and by how much). Instead of point estimates your posterior distributions are parametrized random variables which can be summarized any number of ways. Bayesian tests are also immune to 'peeking' and are thus valid whenever a test is stopped.

The general bayesAB workflow is as follows:

- Decide how you want to parametrize your data (Poisson for counts of email submissions, Bernoulli for CTR on an ad, etc.)
- Use our helper functions to decide on priors for your data (bayesTest, plotDistributions)
- Fit a bayesTest object
  - Optional: Use combine to munge together several bayesTest objects together for an arbitrary / non-analytical target distribution
- `print`, plot.bayesTest, and summary.bayesTest to interpret your results

- Determine whether to stop your test early given the Posterior Expected Loss in `summary` output

Optionally, use banditize and/or deployBandit to turn a pre-calculated (or empty) bayesTest into a multi-armed bandit that can serve recipe recommendations and adapt as new data comes in.

Note, while bayesAB was designed to exploit data related to A/B/etc tests, you can use the package to conduct Bayesian analysis on virtually any vector of data, as long as it can be parametrized by the available functions.

To learn more about bayesAB, start with the vignettes: `browseVignettes(package = "bayesAB")`

`bayesTest` *Fit a Bayesian model to A/B test data.*

Description

This function fits a Bayesian model to your A/B testing sample data. See **Details** for more information on usage.

Usage

```
bayesTest(
  A_data,
  B_data,
  priors,
  n_samples = 1e+05,
  distribution = c("bernoulli", "normal", "lognormal", "poisson", "exponential",
    "uniform", "bernoulliC", "poissonC")
)
```

Arguments

`A_data`	Vector of collected samples from recipe A
`B_data`	Vector of collected samples from recipe B
`priors`	Named vector or named list providing priors as required by the specified distribution:

- For 'bernoulli' distribution `list("alpha" = val1, "beta" = val2)`
- For 'normal' distribution `c("mu" = val1, "lambda" = val2, "alpha" = val3, "beta" = val4)`
- For 'lognormal' distribution `c("mu" = val1, "lambda" = val2, "alpha" = val3, "beta" = val4)`
- For 'poisson' distribution `c("shape" = val1, "rate" = val2)`
- For 'exponential' distribution `list("shape" = val1, "rate" = val2)`
- For 'uniform' distribution `c("xm" = val1, "alpha" = val2)`
- For 'bernoulliC' distribution: same prior definitions as 'bernoulli'
- For 'poissonC' distribution: same prior definitions as 'poisson'

	See plotDistributions or the *Note* section of this help document for more info.
`n_samples`	Number of posterior samples to draw. Should be large enough for the distribution to converge. 1e5 is a good rule of thumb. Not used for closed form tests.
`distribution`	Distribution of underlying A/B test data.

Details

`bayesTest` is the main driver function of the **bayesAB** package. The input takes two vectors of data, corresponding to recipe A and recipe B of an A/B test. Order does not matter, except for interpretability of the final plots and intervals/point estimates. The Bayesian model for each distribution uses conjugate priors which must be specified at the time of invoking the function. Currently, there are *eight* supported distributions for the underlying data:

- Bernoulli: If your data is well modeled by 1s and 0s, according to a specific probability p of a 1 occurring
  - For example, click-through-rate /conversions for a page
  - Data **must** be in a {0, 1} format where 1 corresponds to a 'success' as per the Bernoulli distribution
  - Uses a conjugate `Beta` distribution for the parameter **p** in the Bernoulli distribution
  - `alpha` and `beta` must be set for a prior distribution over **p**
    - alpha = 1, beta = 1 can be used as a diffuse or uniform prior
- Normal: If your data is well modeled by the normal distribution, with parameters μ, σ^2 controlling mean and variance of the underlying distribution
  - Data *can* be negative if it makes sense for your experiment
  - Uses a conjugate `NormalInverseGamma` distribution for the parameters μ and σ^2 in the Normal Distribution.
  - `mu`, `lambda`, `alpha`, and `beta` must be set for prior distributions over μ, σ^2 in accordance with the parameters of the conjugate prior distributions:
    - μ, σ^2 ~ NormalInverseGamma(mu, lambda, alpha, beta)
  - This is a bivariate distribution (commonly used to model mean and variance of the normal distribution). You may want to experiment with both this distribution and the `plotNormal` and `plotInvGamma` outputs separately before arriving at a suitable set of priors for the Normal and LogNormal `bayesTest`

  .
- LogNormal: If your data is well modeled by the log-normal distribution, with parameters μ, σ^2 as the **parameters** of the corresponding log-normal distribution (log of data is ~ N(μ, σ^2))
  - Support for a log-normal distribution is strictly positive
  - The Bayesian model requires same conjugate priors on μ, σ^2 as for the Normal Distribution priors
  - Note: The μ and σ^2 are not the mean/variance of lognormal numbers themselves but are rather the corresponding parameters of the lognormal distribution. Thus, posteriors for the statistics 'Mean' and 'Variance' are returned alongside 'Mu' and 'Sig_Sq' for interpretability.
- Poisson: If your data is well modeled by the Poisson distribution, with parameter λ controlling the average number of events per interval.

  - For example, pageviews per session
  - Data *must* be strictly integral or 0.
  - Uses a conjugate `Gamma` distribution for the parameter λ in the Poisson Distribution
  - `shape` and `rate` must be set for prior distribution over λ
- Exponential: If your data is well modeled by the Exponential distribution, with parameter λ controlling the rate of decay.
  - For example, time spent on a page or customers' LTV
  - Data *must* be strictly >= 0
  - Uses a conjugate `Gamma` distribution for the parameter λ in the Exponential Distribution
  - `shape` and `rate` must be set for prior distribution over λ
- Uniform: If your data is well modeled by the Uniform distribution, with parameter θ controlling the *max* value.
  - bayesAB has only implemented Uniform(0, θ) forms
  - For example, estimating max/total inventory size from individually numbered snapshots
  - Data *must* be strictly > 0
  - Uses a conjugate `Pareto` distribution for the parameter θ in the Uniform(0, θ) Distribution
  - `xm` and `alpha` must be set for prior distribution over θ
- BernoulliC: Closed form (computational) calculation of the 'bernoulli' bayesTest. Same priors are required.
- PoissonC: Closed form (computational) calculation of the 'poisson' bayesTest. Same priors are required.

Value

A `bayesTest` object of the appropriate distribution class.

Note

For 'closed form' tests, you do not get a distribution over the posterior, but simply P(A > B) for the parameter in question.

Choosing priors correctly is very important. Please see http://fportman.com/writing/bayesab-0-dot-7-0-plus-a-primer-on-priors/ for a detailed example of choosing priors within bayesAB. Here are some ways to leverage objective/diffuse (assigning equal probability to all values) priors:

- `Beta`(1, 1)
- `Gamma`(eps, eps) ~ `Gamma`(.00005, .00005) will be effectively diffuse
- `InvGamma`(eps, eps) ~ `InvGamma`(.00005, .00005) will be effectively diffuse
- `Pareto`(eps, eps) ~ `Pareto`(.005, .005) will be effectively diffuse

Keep in mind that the Prior Plots for bayesTest's run with diffuse priors may not plot correctly as they will not be truncated as they approach infinity. See plot.bayesTest for how to turn off the Prior Plots.

Examples

```
A_binom <- rbinom(100, 1, .5)
B_binom <- rbinom(100, 1, .6)

A_norm <- rnorm(100, 6, 1.5)
B_norm <- rnorm(100, 5, 2.5)

AB1 <- bayesTest(A_binom, B_binom,
                 priors = c('alpha' = 1, 'beta' = 1),
                 distribution = 'bernoulli')
AB2 <- bayesTest(A_norm, B_norm,
                 priors = c('mu' = 5, 'lambda' = 1, 'alpha' = 3, 'beta' = 1),
                 distribution = 'normal')

print(AB1)
summary(AB1)
plot(AB1)

summary(AB2)

# Create a new variable that is the probability multiiplied
# by the normally distributed variable (expected value of something)

   AB3 <- combine(AB1, AB2, f = `*`, params = c('Probability', 'Mu'), newName = 'Expectation')

    print(AB3)
    summary(AB3)
    plot(AB3)
```

`c.bayesTest` *Concatenate bayesTest objects*

Description

Concatenate method for objects of class "bayesTest".

Usage

```
## S3 method for class 'bayesTest'
c(..., errorCheck = TRUE)
```

Arguments

`...`	`bayesTest` objects
`errorCheck`	check that objects can be concatenated? Setting this to `FALSE` can have unintended effects if the `bayesTest` objects have different params but can have some speedups if you are sure the objects are of the same type.

Value

A bayesTest object with concatenated posteriors.

Examples

```
A_pois <- rpois(100, 5)
B_pois <- rpois(100, 4.7)

AB1 <- bayesTest(A_pois, B_pois, priors = c('shape' = 25, 'rate' = 5), distribution = 'poisson')
AB2 <- bayesTest(A_pois, B_pois, priors = c('shape' = 25, 'rate' = 5), distribution = 'poisson')
c(AB1, AB2)
```

---

combine *Combine two* bayesAB *objects given a binary function.*

Description

Combine two (or any number, in succession) bayesTest objects into a new arbitrary posterior distribution. The resulting object is of the same class.

Usage

```
combine(bT1, bT2, f = `+`, params, newName)
```

Arguments

bT1	a bayesTest object
bT2	a bayesTest object
f	a binary function (f(x, y)) used to combine posteriors from bT1 to bT2
params	a character vector of length 2, corresponding to names of the posterior parameters you want to combine; defaults to first posterior parameter if not supplied
newName	a string indicating the name of the new 'posterior' in the resulting object; defaults to string representation of f(params[1], params[2])

Value

a bayesTest object with the newly combined posterior samples.

Note

The generics '+.bayesTest', '*.bayesTest', '-.bayesTest', and '/.bayesTest' are shorthand for combine(f = '+'), combine(f = '*'), combine(f = '-'), and combine(f = '/').

See Also

grab

Examples

```
A_binom <- rbinom(100, 1, .5)
B_binom <- rbinom(100, 1, .6)

A_norm <- rnorm(100, 6, 1.5)
B_norm <- rnorm(100, 5, 2.5)

AB1 <- bayesTest(A_binom, B_binom,
                 priors = c('alpha' = 1, 'beta' = 1),
                 distribution = 'bernoulli')

AB2 <- bayesTest(A_norm, B_norm,
                 priors = c('mu' = 5, 'lambda' = 1, 'alpha' = 3, 'beta' = 1),
                 distribution = 'normal')

AB3 <- combine(AB1, AB2, f = `*`, params = c('Probability', 'Mu'), newName = 'Expectation')
# Equivalent to
AB3 <- AB1 * grab(AB2, 'Mu')

# To get the same posterior name as well
AB3 <- rename(AB3, 'Expectation')

# Dummy example
weirdVariable <- (AB1 + AB2) * (AB2 / AB1)
weirdVariable <- rename(weirdVariable, 'confusingParam')

print(AB3)
summary(AB3)
plot(AB3)
```

`deployBandit` *Deploy a bayesBandit object as a JSON API.*

Description

Turn your bayesBandit object into an API and serve/update requests through HTTP.

Usage

```
deployBandit(bandit, port = 8000)
```

Arguments

`bandit`	a bayesBandit object
`port`	port to deploy on

Details

`deployBandit` turns a Bayesian bandit into a JSON API that accepts curl requests. Two of the five methods of bayesBandit classes are exposed: `serveRecipe` and `setResults`. Assuming the API is deployed on `localhost` this is an example of how it would be hit:

$$curl http://localhost:8000/serveRecipe$$

$$curl --data'"A":[1,0,1,1],"B":[0,0,0,1]'http://localhost:8000/setResults$$

Value

An active `http` process on some port.

Examples

```
A_binom <- rbinom(100, 1, .5)
B_binom <- rbinom(100, 1, .6)

AB1 <- bayesTest(A_binom, B_binom, priors = c('alpha' = 1, 'beta' = 1), distribution = 'bernoulli')

binomialBandit <- banditize(AB1)
## Not run: deployBandit(binomialBandit)
```

---

grab *Grab the supplied posterior from a bayesTest object*

Description

Grab the supplied posterior from a bayesTest object, returning another bayesTest object.

Usage

```
grab(bT, posterior)
```

Arguments

`bT`	a bayesTest object
`posterior`	the name of a posterior in that object (string)

Value

a `bayesTest` object with the posterior parameter isolated

See Also

`combine`

plot.bayesTest *Plot bayesTest objects*

Description

Plot method for objects of class "bayesTest".

Usage

```
## S3 method for class 'bayesTest'
plot(
  x,
  percentLift = rep(0, length(x$posteriors)),
  priors = TRUE,
  posteriors = TRUE,
  samples = TRUE,
  ...
)
```

Arguments

x	an object of class "bayesTest"
percentLift	a vector of length(x$posteriors). Each entry corresponds to the percent lift ((A - B) / B) to plot for for the respective posterior in x. Note this is on a 'point' scale. percentLift = 5 implies you want to test for a 5% lift.
priors	logical indicating whether prior plots should be generated.
posteriors	logical indicating whether posterior plots should be generated.
samples	logical indicating whether sample plots should be generated.
...	graphics parameters to be passed to the plotting routines. (For example p, in prior plots)

Note

You can either directly plot a bayesTest object (in which case it will plot interactively), or you can save the plot object to a variable and extract what you need separately. If extracted, you can treat it like any ggplot2 object and modify it accordingly.

Plots are slightly truncated on the extremes to solve the general case of potentially long tail-ends.

Examples

```
A_pois <- rpois(100, 5)
B_pois <- rpois(100, 4.7)

AB1 <- bayesTest(A_pois, B_pois, priors = c('shape' = 25, 'rate' = 5), distribution = 'poisson')

plot(AB1)
```

```
plot(AB1, percentLift = 5)


    p <- plot(AB1)
    p$posteriors$Lambda

## Not run: p$posteriors$Lambda + ggtitle('yolo') # modify ggplot2 object directly
```

`plotBeta` *Plot the PDF of the Beta distribution.*

Description

Plot the PDF of the Beta distribution.

Usage

```
plotBeta(alpha, beta)
```

Arguments

`alpha`	α parameter of the Beta distribution.
`beta`	β parameter of the Beta distribution.

Value

The PDF of Beta(α, β).

Note

The output can be treated like any `ggplot2` object and modified accordingly.

Examples

```
plotBeta(1, 1)
plotBeta(2, 5)
## Not run: plotBeta(2, 5) + ggtitle('I hate the default title!')
```

plotDistributions *Plot distributions to explore data and/or choose priors.*

Description

These are helper functions to help explore data and/or choose priors. Click further for more details.

- plotBeta
- plotGamma
- plotInvGamma
- plotLogNormal
- plotNormal
- plotPareto
- plotPoisson

Note

Choosing priors correctly is very important. Please see http://fportman.com/writing/bayesab-0-dot-7-0-plus-a-primer-on-priors/ for a detailed example of choosing priors within bayesAB. Here are some ways to leverage objective/diffuse (assigning equal probability to all values) priors:

- `Beta`(1, 1)
- `Gamma`(eps, eps) ~ `Gamma`(.00005, .00005) will be effectively diffuse
- `InvGamma`(eps, eps) ~ `InvGamma`(.00005, .00005) will be effectively diffuse
- `Pareto`(eps, eps) ~ `Pareto`(.005, .005) will be effectively diffuse

Keep in mind that the Prior Plots for bayesTest's run with diffuse priors may not plot correctly as they will not be truncated as they approach infinity. See plot.bayesTest for how to turn off the Prior Plots.

plot... functions are generated programmatically so the function calls in their body will be substituted directly

plotGamma *Plot the PDF of the Gamma distribution.*

Description

Plot the PDF of the Gamma distribution.

Usage

```
plotGamma(shape, rate)
```

Arguments

shape	shape (α) parameter of the Gamma distribution.
rate	rate (β) parameter of the Gamma distribution.

Details

Note: We use the shape/rate parametrization of Gamma. See https://en.wikipedia.org/wiki/Gamma_distribution for details.

Value

The PDF of Gamma(shape, rate).

Note

The output can be treated like any ggplot2 object and modified accordingly.

Examples

```
plotGamma(1, 1)
plotGamma(2, 5)
## Not run: plotGamma(2, 5) + ggtitle('I hate the default title!')
```

---

plotInvGamma *Plot the PDF of the Inverse Gamma distribution.*

Description

Plot the PDF of the Inverse Gamma distribution.

Usage

```
plotInvGamma(shape, scale)
```

Arguments

shape	shape parameter of the Inverse Gamma distribution.
scale	scale parameter of the Inverse Gamma distribution.

Value

The PDF of InvGamma(shape, scale).

Note

The output can be treated like any ggplot2 object and modified accordingly. Also note that the scale parameter of the Inverse Gamma distribution is analogous to the beta (or rate) parameter of the regular Gamma distribution. The beta parameter of the plotNormalInvGamma distribution is analogous to the scale parameter here.

Examples

```
plotInvGamma(2, 4)
plotInvGamma(1, 17)
## Not run: plotInvGamma(1, 17) + ggtitle('I hate the default title!')
```

`plotLogNormal` *Plot the PDF of the Log Normal distribution.*

Description

Plot the PDF of the Log Normal distribution.

Usage

```
plotLogNormal(mu, sigma)
```

Arguments

`mu`	μ parameter of the Log Normal distribution.
`sigma`	σ parameter of the Log Normal distribution.

Value

The PDF of Log Normal(μ, σ^2).

Note

The output can be treated like any `ggplot2` object and modified accordingly.

Examples

```
plotLogNormal(1, 1)
plotLogNormal(2, .5)
## Not run: plotLogNormal(2, .5) + ggtitle('I hate the default title!')
```

plotNormal *Plot the PDF of the Normal distribution.*

Description

Plot the PDF of the Normal distribution.

Usage

```
plotNormal(mu, sd)
```

Arguments

mu	μ parameter of the Normal distribution.
sd	σ parameter of the Normal distribution.

Value

The PDF of Normal(μ, σ^2).

Note

The output can be treated like any ggplot2 object and modified accordingly.

Examples

```
plotNormal(1, 1)
plotNormal(2, 5)
## Not run: plotNormal(2, 5) + ggtitle('I hate the default title!')
```

plotNormalInvGamma *Plot the bivariate PDF of the Normal Inverse Gamma Distribution.*

Description

Plot the bivariate PDF of the Normal Inverse Gamma Distribution.

Usage

```
plotNormalInvGamma(mu, lambda, alpha, beta)
```

Arguments

mu	μ parameter of the Normal Inverse Gamma distribution.
lambda	λ parameter of the Normal Inverse Gamma distribution.
alpha	α parameter of the Normal Inverse Gamma distribution.
beta	β parameter of the Normal Inverse Gamma distribution.

Value

The PDF of NormalInverseGamma(mu, lambda, alpha, beta)

Note

This is a bivariate distribution (commonly used to model mean and variance of the normal distribution) and returns a 2d contour plot instead of a typical one dimensional PDF. You may want to experiment with both this distribution and the `plotNormal` and `plotInvGamma` outputs separately before arriving at a suitable set of priors for the Normal and LogNormal `bayesTest`.

Examples

```
plotNormalInvGamma(3, 1, 1, 1)
```

`plotPareto` *Plot the PDF of the Pareto distribution.*

Description

Plot the PDF of the Pareto distribution.

Usage

```
plotPareto(xm, alpha)
```

Arguments

`xm`	xm parameter of the Pareto distribution.
`alpha`	alpha parameter of the Pareto distribution.

Value

The PDF of Pareto(xm, alpha).

Note

The output can be treated like any `ggplot2` object and modified accordingly.

Examples

```
plotPareto(1, 1)
plotPareto(5, 3)
## Not run: plotPareto(5, 3) + ggtitle('I hate the default title!')
```

plotPoisson *Plot the PDF of the Poisson distribution.*

Description

Plot the PDF of the Poisson distribution.

Usage

```
plotPoisson(lambda)
```

Arguments

lambda	λ parameter of the Poisson distribution.

Value

The PDF of Poisson(λ).

Note

The output can be treated like any ggplot2 object and modified accordingly.

Examples

```
plotPoisson(1)
plotPoisson(5)
## Not run: plotPoisson(5) + ggtitle('I hate the default title!')
```

rename *Rename the posterior for a bayesTest object*

Description

Rename the posterior param for a bayesTest object.

Usage

```
rename(bT, newName)
```

Arguments

bT	a bayesTest object
newName	the new name you want for the posterior param (string)

Value

a bayesTest object with the posterior parameter renamed

See Also

combine

---

summary.bayesTest *Summarize bayesTest objects*

Description

Summary method for objects of class "bayesTest".

Usage

```
## S3 method for class 'bayesTest'
summary(
  object,
  percentLift = rep(0, length(object$posteriors)),
  credInt = rep(0.9, length(object$posteriors)),
  ...
)
```

Arguments

object	an object of class "bayesTest"
percentLift	a vector of length(x$posteriors). Each entry corresponds to the percent lift ((A - B) / B) to summarize for for the respective posterior in x. Note this is on a 'point' scale. percentLift = 5 implies you want to test for a 5% lift.
credInt	a vector of length(x$posteriors). Each entry corresponds to the width of credible interval of (A - B) / B to calculate for the respective posterior in x. Also on a 'point' scale.
...	additional arguments affecting the summary produced.

Value

A summaryBayesTest object which contains summaries of the Posterior distributions, direct probabilities that A > B (by percentLift), credible intervals on (A - B) / B, and the Posterior Expected Loss on all estimated parameters.

Note

The Posterior Expected Loss (https://en.wikipedia.org/wiki/Bayes_estimator) is a good indicator of when to end a Bayesian AB test. If the PEL is lower than the absolute delta of the minimum effect you wish to detect, the test can be reasonably be stopped.

Examples

```
A_pois <- rpois(100, 5)
B_pois <- rpois(100, 4.7)

AB1 <- bayesTest(A_pois, B_pois, priors = c('shape' = 25, 'rate' = 5), distribution = 'poisson')

summary(AB1)
summary(AB1, percentLift = 10, credInt = .95)
```

Index