Fast Bayesian A/B and multivariate testing

Guillermo Navas Palencia

BBVA, Global Risk Management – Analytics

PyDay BCN 2019

Barcelona, 16^{th} November 2019

Table of Contents

Introduction (1 / 2)

- Frequentist inference (classical inference / classical point estimation)
  - Base on asymptotic performance $\Longrightarrow$ Central Limit Theorem (CLT).
  - P-value: the probability of seeing a result at least as *extreme* as a real result after a A/A test of the same size [Stu15]. It is defined as

$$\text{p-value} = P[t \geq t_e | H_0], \quad H_0 : B \equiv A$$

  - Hypothesis testing based on rejecting H_0. p-value $\neq P[B > A]$!
  - Confidence interval: if we repeated the same experiment used to construct an interval for an unobserved value n times ($n \to \infty$), $(100 - \delta)\%$ of the intervals would contain the true value. This is not a credible interval!
  - Set and fix stopping rule and sample size (power test calculation).
- Bayesian inference
  - Bayes Theorem: given a prior distribution $p(\theta)$, update belief based on sample x

$$p(\theta|x) = \frac{f(x|\theta)p(\theta)}{f(x)}$$

  - Choose of prior parameters $\Longrightarrow$ update $\Longrightarrow$ posterior parameters.
  - Calculation of posterior distribution.
  - Calculation of predictive posterior distribution.

Introduction (2 / 2)

- Bayesian inference: advantages
  - Ease interpretability.
  - Sample size is not fixed in advance $\Longrightarrow$ repeated/streaming testing.
  - Account for uncertainty; points estimates $\Longrightarrow$ random variables.
  - Immune to data peeking
- Bayesian inference: disadvantages
  - Analytical tractability
  - Computational cost

Bayesian testing metrics: probability to beat

- **A/B testing**: the *error probability or probability* of $X_B > X_A$

$$E(B) = P[X_B > X_A]$$

```
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>> (xb > xa).mean()
```

- **Multivariate testing**: the *probability to beat all*

$$E(X_i) = P\left[X_i > \max_{j \neq i} X_j\right]$$

```
>>> import numpy as np
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>> xc = stats.beta(5, 60).rvs(size=int(1e6))
>>> xd = stats.beta(7, 90).rvs(size=int(1e6))
>>> maxall = np.maximum.reduce([xa, xc, xd])
>>> (xb > maxall).mean()
```

Bayesian testing metrics: expected loss

- **A/B testing**: the expected value of the *loss function*

$$EL(B) = \mathrm{E}[\max(X_A - X_B, 0)]$$

```
>>> import numpy as np
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>> np.maximum(xa - xb, 0).mean()
```

- **Multivariate testing**: the *expected loss function vs all*

$$EL(X_i) = \mathrm{E}[\max(\max_{j \neq i} X_j - X_i, 0)]$$

```
>>> import numpy as np
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>> xc = stats.beta(5, 60).rvs(size=int(1e6))
>>> xd = stats.beta(7, 90).rvs(size=int(1e6))
>>> maxall = np.maximum.reduce([xa, xc, xd])
>>> np.maximum(maxall - xb, 0).mean()
```

Bayesian testing metrics: expected relative loss

- **A/B testing**: the expected value of the relative *loss function*

$$ERL(B) = \mathrm{E}[(X_A - X_B)/X_B]$$

```
>>> import numpy as np
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>>((xa - xb) / xb).mean()
```

- **Multivariate testing**: the *expected relative loss function vs all*

$$ERL(X_i) = \mathrm{E}[(\max_{j \neq i} X_j - X_i)/X_i]$$

```
>>> import numpy as np
>>> from scipy import stats
>>> xa = stats.beta(2, 10).rvs(size=int(1e6))
>>> xb = stats.beta(3, 12).rvs(size=int(1e6))
>>> xc = stats.beta(5, 60).rvs(size=int(1e6))
>>> xd = stats.beta(7, 90).rvs(size=int(1e6))
>>> maxall = np.maximum.reduce([xa, xc, xd])
>>> ((maxall - xb) / xb).mean()
```

Computation of credible intervals

Definition: A credible interval is a region with a particular probability to contain an unobserved value. Bayesian equivalent of the confidence interval.
Given a significance level δ:

- **Equally-tailed Interval (ETI)**: Credible interval using the quantile method, with quantile function $Q = F^{-1}$, solving $F(z) = \delta/2$ and $F(z) = 1 - \delta/2$, satisfying

$$P(Q(\delta/2) < Z < Q(1 - \delta/2)) = 1 - \delta$$

  - Assumption: distribution is symmetric.
- **Highest Density Interval (HDI)**: Solving

$$P(l < Z < u) = 1 - \delta$$

  for l and u, being the lower and upper bound of the interval.
  - No assumptions, appropriate for symmetric and skewed distributions.

Computation of credible intervals: HDI – Monte Carlo sampling

The HDI computes the narrowest of the infinite intervals satisfying $P(l < Z < u) = 1 - \delta$. R code in [Kru15]. `NumPy` implementation to compute HDI given MC samples

```
>>> import numpy as np
>>> n = len(x)
>>> xsorted = np.sort(x)
>>> n_included = int(np.ceil(interval_length * n))
>>> n_ci = n - n_included
>>> ci = xsorted[n_included:] - xsorted[:n_ci]
>>> j = np.argmin(ci)
>>> hdi_min = xsorted[j]
>>> hdi_max = xsorted[j + n_included]
```

Computation of credible intervals: HDI – mathematical optimization

The HDI computes the narrowest interval by solving the minimization problem [CS99],

$$\min_{l<u} \left(|f(u) - f(l)| + |F(u) - F(l) - (1-\delta)| \right).$$

Reformulation: remove absolute values and add term $u - l$

$$\begin{aligned}
\min_{u,l,t,w} \quad & t + w + u - l \\
\text{s.t.} \quad & -t + f(u) - f(l) \geq 0 \\
& t + f(u) - f(l) \geq 0 \\
& -w + F(u) - F(l) - (1-\delta) \geq 0 \\
& w + F(u) - F(l) - (1-\delta) \geq 0 \\
& u - l - \epsilon \geq 0 \\
& l \in [l_{\min}, l_{\max}]] \\
& u \in [u_{\min}, u_{\max}]
\end{aligned}$$

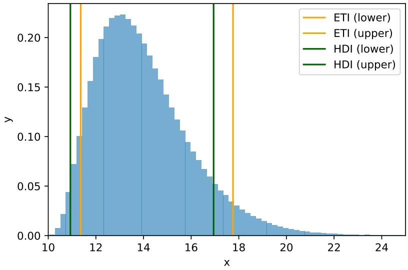
where $\epsilon > 0$. Parameters $l_{\min}$, $l_{\max}$, $u_{\min}$ and $u_{\max}$ denote the bounds for the interval limits l and u.

Computation of credible intervals: `scipy.optimize`

```
def func(x):
    return x[3] + x[2] + x[1] - x[0]

def obj_f(x):
    return f.pdf(x[1]) - f.pdf(x[0])

def obj_F(x):
    return f.cdf(x[1]) - f.cdf(x[0])

epsilon = 1e-6

cons = (
    {'type': 'ineq', 'fun': lambda x: x[1] - x[0] - epsilon},
    {'type': 'ineq', 'fun': lambda x: -x[2] + obj_f(x)},
    {'type': 'ineq', 'fun': lambda x: x[2] + obj_f(x)},
    {'type': 'ineq', 'fun': lambda x: -x[3] + obj_F(x) - interval_length},
    {'type': 'ineq', 'fun': lambda x: x[3] + obj_F(x) - interval_length}
)

res = optimize.minimize(func, (*x0, 0, 0), method="SLSQP",
                        constraints=cons, bounds=[*bounds, (0, 1), (0, 1)])
```

Computation of credible intervals: example 1

```
>>> from scipy import stats
>>> from cprior.cdist import ci_interval
>>> x = stats.gamma(4, 10).rvs(size=int(1e6), random_state=42)
>>> ci_interval(x=x, interval_length=0.9, method="ETI")
array([11.36321512, 17.75748775])
>>> ci_interval(x=x, interval_length=0.9, method="HDI")
array([10.92933934, 16.94237247])
```

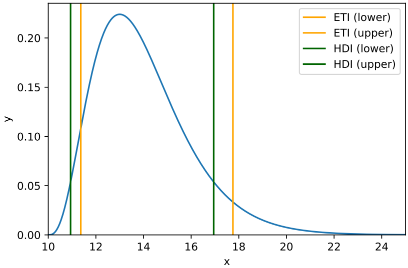


Timings (`%timeit`)[1]: ETI: 18 ms, HDI: 107 ms

[1] Intel(R) Core(TM) i5-3317 CPU at 1.70GHz.

Computation of credible intervals: example 2

```
>>> import numpy as np
>>> from scipy import stats
>>> from cprior.cdist import ci_interval
>>> dist = stats.gamma(4, 10)
>>> ci_interval_exact(dist=dist, interval_length=0.9, method="ETI")
array([11.3663184 , 17.75365653])
>>> bounds = [(0, np.inf), (0, np.inf)]
>>> ci_interval_exact(dist=dist, interval_length=0.9, method="HDI", bounds=bounds)
array([10.93729501, 16.94611345])
```



Timings (`%timeit`): ETI: 0.2 ms, HDI: 45 ms

The CPrior library

- Python/C++ library, open source (LGPL-3.0)
- Github: `https://github.com/guillermo-navas-palencia/cprior`
- Documentation: `http://gnpalencia.org/cprior/`
- Technical notes [NP19]
- Support several conjugate prior distributions
  - Beta distribution ✓
  - Gamma distribution ✓
  - Pareto distribution ✓
  - Normal-inverse-gamma distribution ✓
  - Others: beta-binomial, inverse gamma, multivariate distributions...
- Fast and accurate results:
  - Development of closed-forms in terms of special functions
  - Fast Monte Carlo methods
    - Median Latin Hypercube Sampling
    - Parallel crude Monte Carlo
  - Numerical integration
  - *Streaming Bayesian testing*
- $\sim$15000 lines of code

CPrior testing metrics: probability to beat

Let us consider probability distributions X_i with support $\mathbb{R}$.

- **A/B testing**: the *error probability or probability* of $X_B > X_A$

$$P[X_B > X_A] = \int_{-\infty}^{\infty} \int_{x_A}^{\infty} f(x_A, x_B)\, dx_B\, dx_A,$$

where $f(x_A, x_B)$ is the joint probability distribution, under the assumption of independence, i.e. $f(x_A, x_B) = f(x_A)f(x_B)$.

- **Multivariate testing**: the *probability to beat all*

$$P\left[X_i > \max_{j \neq i} X_j\right] = \int_{-\infty}^{\infty} f(x_i) \prod_{j \neq i} F_{X_j}(x_i)\, dx_i\,.$$

Given $X_{max} = \max\{X_1, \ldots, X_n\}$. The cumulative distribution function is

$$F_{X_{max}}(z) = P\left[\max_{i=1,\ldots,n} X_i \leq z\right] = \prod_{i=1}^{n} P[X_i \leq z] = \prod_{i=1}^{n} F_{X_i}(z),$$

where $F_{X_i}(z)$ is the cdf of each random variable X_i.

CPrior testing metrics: expected loss

Let us consider probability distributions X_i with support $\mathbb{R}$.

- **A/B testing**: the *expected loss function* if variant X_B is chosen

$$EL(X_B) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \max(x_A - x_B, 0) f(x_A, x_B)\, dx_B\, dx_A\,.$$

- **Multivariate testing**: the *expected loss function vs all*, taking $Y = \max\limits_{j \neq i} X_j$

$$EL(X_i) = \int_{-\infty}^{\infty} y f(y) F_{X_i}(y)\, dy - \int_{-\infty}^{\infty} f(y) F^*_{X_i}(y)\, dy,$$

where $F^*_{X_i}(y) = \int_{-\infty}^{y} x_i f(x_i)\, dx_i$. The probability density function is obtain after derivation of $F_{X_{max}}(z)$

$$f_{X_{max}}(z) = \sum_{i=1}^{n} f_{X_i}(z) \prod_{j \neq i} F_{X_j}(z),$$

where $f_{X_i}(z)$ is the pdf of each random variable X_i.

Beta distribution: A/B testing

- Probability to beat: given two distributions $X_A \sim \mathcal{B}(\alpha_A, \beta_A)$ and $X_B \sim \mathcal{B}(\alpha_B, \beta_B)$, $P[X_B > X_A]$ is given by[2]

$$P[X_B > X_A] = 1 - \frac{B(\alpha_A + \alpha_B, \beta_A)}{\alpha_B B(\alpha_A, \beta_A) B(\alpha_B, \beta_B)} \, {}_3F_2 \left(\begin{matrix} \alpha_B, \alpha_A + \alpha_B, 1 - \beta_B \\ 1 + \alpha_B, \alpha_A + \alpha_B + \beta_A \end{matrix} ; 1 \right),$$

  where $B(a, b)$ is the beta function and ${}_3F_2(a, b, c; d, e; z)$ is the generalized hypergeometric function.
- Implementation hypergeometric series (C++) `https://github.com/guillermo-navas-palencia/cprior/blob/master/cprior/_lib/src/beta.cpp`
  - Special cases in terms of the regularized incomplete beta function $I_x(a, b)$.
- Timings

```
%timeit abtest.probability(variant="B", method="exact")
```

  10.2 μs $\pm$ 58.2 ns per loop (mean $\pm$ std. dev. of 7 runs, 100000 loops each)

[2] `http://gnpalencia.org/cprior/formulas_conjugate_beta.html`

Beta distribution: Multivariate testing - MLHS

- Probability to beat all:

$$P\left[X_i > \max_{j \neq i} X_j\right] = \int_0^1 \frac{x^{\alpha_i - 1}(1-x)^{\beta_i - 1}}{B(\alpha_i, \beta_i)} \prod_{j \neq i} I_x(\alpha_j, \beta_j)\, dx$$

$$= \mathrm{E}\left[\prod_{j \neq i} I_X(\alpha_j, \beta_j)\right], \quad X \sim \mathcal{B}(\alpha_i, \beta_i).$$

- Median Latin Hypercube Sampling (MLHS)

1. $a \leftarrow 0$, $b \leftarrow 1$
2. vector of indexes: π_i, $i = 1, \ldots, n$
3. random shuffle of π_i
4. $v_i = (b - a)\frac{\pi_i - 0.5}{n} + a$
5. $x_i = F^{-1}(v_i)$

```
r = np.arange(1, mlhs_samples + 1)
np.random.shuffle(r)
v = (r - 0.5) / mlhs_samples
x = self.models[variant].ppf(v)
```

```
np.mean(np.prod([special.betainc(a, b, x)
        for a, b in variant_params], axis=0))
```

Beta distribution: Multivariate testing - numerical integration

- Probability to beat all:

$$P\left[X_i > \max_{j \neq i} X_j\right] = \int_0^1 \frac{x^{\alpha_i - 1}(1-x)^{\beta_i - 1}}{B(\alpha_i, \beta_i)} \prod_{j \neq i} I_x(\alpha_j, \beta_j)\, dx\,.$$

```
def func_mv_prob(x, a, b, variant_params):
    pdf = (a - 1) * np.log(x) + (b - 1) * np.log(1 - x) - special.betaln(a, b)
    g = np.prod([special.betainc(a, b, x) for a, b in variant_params], axis=0)
    return np.exp(pdf) * g

integrate.quad(func=func_mv_prob, a=0, b=1, args=(a, b, variant_params))[0]
```

- Benchmark (5 variants)

Method	Samples	Rel. error	time
Monte Carlo	1e4	8e-2	50 ms
	1e5	1e-2	137 ms
	1e6	2e-3	1160 ms
MLHS	1e2	3e-3	2 ms
	1e3	3e-4	8 ms
	1e4	3e-5	65 ms
Quad	-	-	26 ms

Gamma distribution: A/B testing

- Probability to beat: given two distributions $X_A \sim \mathcal{G}(\alpha_A, \beta_A)$ and $X_B \sim \mathcal{G}(\alpha_B, \beta_B)$, $P[X_B > X_A]$ is given by[3]

$$
\begin{aligned}
P[X_B > X_A] &= 1 - \frac{\beta_A^{\alpha_A} \beta_B^{\alpha_B}}{(\beta_A + \beta_B)^{\alpha_A + \alpha_B}} \frac{{}_2F_1\left(1, \alpha_A + \alpha_B; \alpha_B + 1; \frac{\beta_B}{\beta_B + \beta_A}\right)}{\alpha_B B(\alpha_A, \alpha_B)} \\
&= I_{\frac{\beta_A}{\beta_A + \beta_B}}(\alpha_A, \alpha_B),
\end{aligned}
$$

  where ${}_2F_1(a, b; c; z)$ is the Gauss hypergeometric function and $I_x(a, b)$ is the regularized incomplete beta function.

- Expected loss:

$$
EL(X_B) = \frac{\alpha_A}{\beta_A} I_{\frac{\beta_B}{\beta_A + \beta_A}}(\alpha_B, \alpha_A + 1) - \frac{\alpha_B}{\beta_B} I_{\frac{\beta_B}{\beta_A + \beta_A}}(\alpha_B + 1, \alpha_A).
$$

- Implementation $I_x(a, b)$: `scipy.special.betainc`.

[3] `http://gnpalencia.org/cprior/formulas_conjugate_gamma.html`

Gamma distribution: Multivariate testing - MLHS

- Expected loss vs all:

$$EL(X_i) = \mathrm{E}\left[YP(\alpha_i, \beta_i Y) - \frac{\alpha_i}{\beta_i} P(\alpha_i + 1, \beta_i Y)\right], \quad Y \sim \max_{j \neq i} \mathcal{G}(\alpha_j, \beta_j),$$

where $P(\alpha, \beta)$ is the regularized lower incomplete gamma function.

```
r = np.arange(1, mlhs_samples + 1)
np.random.shuffle(r)
v = (r - 0.5) / mlhs_samples
x = np.array([optimize.brentq(f=func_mv_ppf, args=(variant_params, p),
              a=0, b=n, xtol=1e-4, rtol=1e-4) for p in v])

p = x * special.gammainc(a, b * x)
q = a / b * special.gammainc(a + 1, b * x)
np.mean(p - q)
```

- Benchmark (5 variants)

Method	Samples	Rel. error	time
MC	1e4	2e-3	37 ms
	1e5	2e-4	100 ms
MLHS	1e2	9e-3	31 ms
	1e3	1e-3	255 ms
Quad	-	-	54 ms

Gamma distribution: Multivariate testing - MLHS

- Expected relative loss vs all:

$$ELR(X_i) = \mathrm{E}\left[Y\right] \frac{\beta_i}{\alpha_i - 1} - 1, \quad Y \sim \max_{j \neq i} \mathcal{G}(\alpha_j, \beta_j).$$

```
r = np.arange(1, mlhs_samples + 1)
np.random.shuffle(r)
v = (r - 0.5) / mlhs_samples
v = v[..., np.newaxis]
n = len(variant_params)
aa, bb = map(np.array, zip(*variant_params))
cc = aa / bb
xx = stats.gamma(a=aa + 1, loc=0, scale=1.0 / bb).ppf(v)

return np.sum([cc[i] * np.prod([
              special.gammainc(aa[j], bb[j] * xx[:, i])
              for j in range(n) if j != i], axis=0)
              for i in range(n)], axis=0).mean()
```

- Benchmark (5 variants)

Method	Samples	Rel. error	time
MC	1e4	4e-3	35 ms
	1e5	5e-4	90 ms
MLHS	1e2	6e-3	4 ms
	1e3	8e-4	16 ms
Quad	-	-	48 ms

Bayesian experiment: Bernoulli distribution (1 / 5)

A Bayesian multivariate test with control and 3 variants. Data follows a Bernoulli distribution with distinct success probability.

- Generate control and variant models and build experiment. Select stopping rule and threshold (epsilon).

```
from scipy import stats
from cprior.models.bernoulli import BernoulliModel
from cprior.models.bernoulli import BernoulliMVTest
from cprior.experiment.base import Experiment

modelA = BernoulliModel(name="control", alpha=1, beta=1)
modelB = BernoulliModel(name="variation", alpha=1, beta=1)
modelC = BernoulliModel(name="variation", alpha=1, beta=1)
modelD = BernoulliModel(name="variation", alpha=1, beta=1)

mvtest = BernoulliMVTest({"A": modelA, "B": modelB, "C": modelC, "D": modelD})

experiment = Experiment(name="CTR", test=mvtest,
                        stopping_rule="probability_vs_all",
                        epsilon=0.99, min_n_samples=1000, max_n_samples=None)
```

Bayesian experiment: Bernoulli distribution (2 / 5)

- Check experiment description.

```
>>> experiment.describe()

=====================================================
  Experiment: CTR
=====================================================
    Bayesian model:                        bernoulli-beta
    Number of variants:                                 4

    Options:
      stopping rule                   probability_vs_all
      epsilon                                    0.99000
      min_n_samples                                 1000
      max_n_samples                              not set

    Priors:

         alpha  beta
      A      1     1
      B      1     1
      C      1     1
      D      1     1
  -------------------------------------------------
```

Bayesian experiment: Bernoulli distribution (3 / 5)

- Generate or pass new data and update models until a clear winner is found.
- The stopping rule will be updated after a new update.

```
with experiment as e:
    while not e.termination:
        data_A = stats.bernoulli(p=0.0223).rvs(size=25)
        data_B = stats.bernoulli(p=0.1128).rvs(size=15)
        data_C = stats.bernoulli(p=0.0751).rvs(size=35)
        data_D = stats.bernoulli(p=0.0280).rvs(size=15)

        e.run_update(**{"A": data_A, "B": data_B, "C": data_C, "D": data_D})

    print(e.termination, e.status)

True winner B
```

Bayesian experiment: Bernoulli distribution (4 / 5)

- Reporting: experiment summary.

```
>>> experiment.summary()
```

	name	probability	expected_loss	improvement	probability_vs_all	expected_loss_vs_all	improvement_vs_all	n_samples
A	control	-	-	-	0.00%	0.0881716	-572.15%	1675
B	variation	100.00%	1.30573e-27	84.43%	99.94%	1.63007e-06	32.00%	1005
C	variation	100.00%	5.95894e-21	76.97%	0.06%	0.0339692	-49.16%	2345
D	variation	97.89%	4.26579e-05	40.01%	0.00%	0.0764664	-288.51%	1005

- Reporting: statistics collected data throughout the experiment.

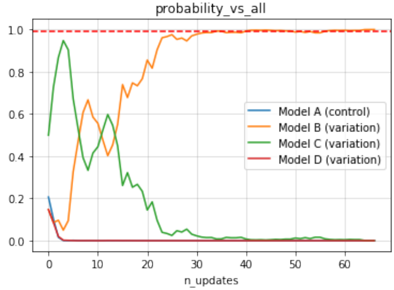
```
>>> experiment.stats()
```

	A	B	C	D
count	1675.000000	1005.000000	2345.000000	1005.000000
mean	0.019104	0.111443	0.073774	0.028856
std	0.136933	0.314836	0.261458	0.167484
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

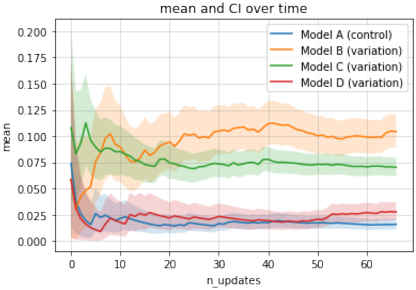
Bayesian experiment: Bernoulli distribution (5 / 5)

- Reporting: visualize stopping rule metric over time (updates).
- Reporting: visualize statistics over time (updates).

```
>>> experiment.plot_metric()
```




```
>>> experiment.plot_stats()
```



Bayesian experiment: normal distribution (1 / 4)

A Bayesian multivariate test with control and 3 variants. Data follows a normal distribution with distinct mean and standard deviation.

- Generate control and variant models and build experiment. Select stopping rule and threshold (epsilon).

```
from scipy import stats
from cprior.models import NormalModel
from cprior.models import NormalMVTest
from cprior.experiment.base import Experiment

modelA = NormalModel(name="control")
modelB = NormalModel(name="variation")
modelC = NormalModel(name="variation")
modelD = NormalModel(name="variation")

mvtest = NormalMVTest({"A": modelA, "B": modelB, "C": modelC, "D": modelD})

experiment = Experiment(name="GPA", test=mvtest,
                        stopping_rule="probability_vs_all", epsilon=0.99,
                        min_n_samples=500, max_n_samples=None,
                        nig_metric="mu")
```

Bayesian experiment: normal distribution (2 / 4)

- Check experiment description.

```
>>> experiment.describe()

=====================================================
  Experiment: GPA
=====================================================
    Bayesian model:     normal-normalinversegamma
    Number of variants:                             4

    Options:
      stopping rule              probability_vs_all
      epsilon                                 0.99000
      min_n_samples                               500
      max_n_samples                           not set

    Priors:

           loc  variance_scale  shape  scale
      A  0.001           0.001  0.001  0.001
      B  0.001           0.001  0.001  0.001
      C  0.001           0.001  0.001  0.001
      D  0.001           0.001  0.001  0.001
  -------------------------------------------------
```

Bayesian experiment: normal distribution (3 / 4)

- Generate or pass new data and update models until a clear winner is found.
- The stopping rule will be updated after a new update.

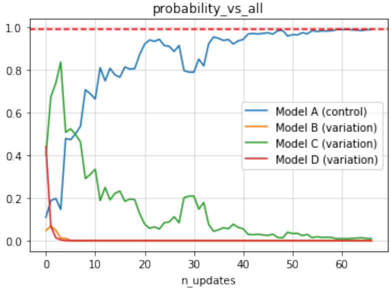
```
with experiment as e:
    while not e.termination:
        data_A = stats.norm(loc=8, scale=3).rvs(size=10)
        data_B = stats.norm(loc=7, scale=2).rvs(size=25)
        data_C = stats.norm(loc=7.5, scale=4).rvs(size=12)
        data_D = stats.norm(loc=6.75, scale=2).rvs(size=8)

        e.run_update(**{"A": data_A, "B": data_B, "C": data_C, "D": data_D})

    print(e.termination, e.status)
```

```
True winner A
```

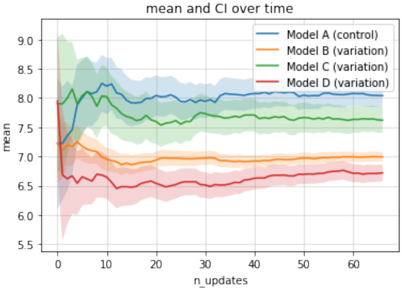
Bayesian experiment: normal distribution (4 / 4)

- Reporting: visualize stopping rule metric over time (updates).
- Reporting: visualize statistics over time (updates).

```
>>> experiment.plot_metric()
```




```
>>> experiment.plot_stats()
```



Bibliography

M. Chen and Q. Shao.
Monte Carlo Estimation of Bayesian Credible and HPD Intervals.
Journal of Computational and Graphical Statistics, 8(1):69–92, 1999.

J. K. Kruschke.
Doing Bayesian Data Analysis: A Tutorial with R, JAGS and Stan.
Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2015.

G. Navas-Palencia.
CPrior: Technical notes, 2019.
`http://gnpalencia.org/cprior/formulas_models.html`.

C. Stucchio.
Bayesian A/B Testing at VWO.
Visual Web Optimizer, 2015.

Thank you!

- https://github.com/guillermo-navas-palencia/cprior