Bayesian Inference for Fun and Profit

Mitzi Morris

Stan Development Team

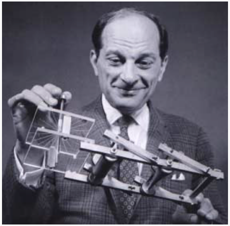
Columbia University, New York NY

November 6, 2019

Bayesian Inference - talk outline

- General remarks and review
- Multi-level regression models in Stan
- Evaluating your inference

Stan - *the man, the language, the software*



- Named after *Stanislaw Ulam*
  originator of Monte Carlo estimation
- Probabilistic programming language
- Stan NUTS-HMC sampler
  Markov Chain Monte Carlo (MCMC) sampler
  - PyMC3 and Pyro also use NUTS-HMC
- Rich eco-system of downstream analysis packages (in R)
  - Arviz!
- Open-source - https://github.com/stan-dev
  - CmdStanPy is BSD licensed

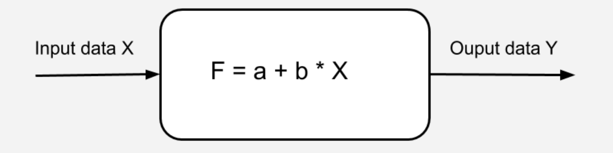
Multi-level modeling in Stan

- Multi-level modeling is a generalization of regression modeling.
- Stan was developed in order to fit complex multi-level models.
- *Information pooling* (what ML calls "regularization")
- Also known as:
  - *hierarchical (generalized) linear models, nested data models, mixed models, random coefficients, random-effects, random parameter models, split-plot designs*, ...
  - see: "All the names for hierarchical and multilevel modeling" blog "Statistical Modeling, Causal Inference, and Social Science"

Statistical Inference

We learn about unknown or unobserved quantities of a process from the data generated by that process.

The **parametric model** is a **white box model**



We work backwards from the generated data (outputs) to find out what's in the box!

Bayesian Workflow

- **Modelling**: define a model (approximate) of the *data generating process*.
- **Estimation**: determine the *posterior probability* of the model parameters conditional on the data.
- **Model Checking**: evaluate how well the model fits the data.
- **Model Improvement**: Iterate steps 1-3.
- **Model Comparison**
    - are results reasonable?
    - how sensitive are results to model assumptions?

Statistical Notation

- y - data
- θ - parameters
- p(θ) - **prior probability distribution** - the probability of the parameters before any data are observed
- p(y, θ) - **joint probability distribution** of the data and parameters
- p$(\theta|\, y)$ - **posterior probability distribution** - the probability of the parameters conditional on the data
- p$(y|\, \theta)$ - *probability of the data given the parameters*
  - if y is fixed, this is the **likelihood function**
  - if θ is fixed, this is the **sampling distribution**

Bayesian Inference

Bayes' Theorem relates the **conditional probability** of the parameters given the data, $\mathrm{p}(\theta|y)$, to the **joint probability** of parameters and data, $\mathrm{p}(\theta, y)$.

$$\begin{aligned}
\mathrm{p}(\theta|\mathrm{y}) &= \frac{\mathrm{p}(\mathrm{y},\theta)}{\mathrm{p}(\mathrm{y})} && \text{[def of conditional probability]} \\
&= \frac{\mathrm{p}(\mathrm{y}|\theta)\,\mathrm{p}(\theta)}{\mathrm{p}(\mathrm{y})} && \text{[rewrite joint probability as conditional]}
\end{aligned}$$

$p(y)$ doesn't depend on θ - proportional constant for fixed y

can be omitted - all we need to compute is:

$$\mathrm{p}(\theta|\mathrm{y}) \propto \mathrm{p}(\mathrm{y}|\theta)\,\mathrm{p}(\theta) \qquad \text{[unnormalized posterior density]}$$

The posterior is **proportional** to the **prior** times the **likelihood**

Bayesian Inference

The posterior is **proportional** to the **prior** times the **likelihood**

$$\mathrm{p}(\theta|\mathrm{y}) \quad \propto \quad \mathrm{p}(\mathrm{y}|\theta)\,\mathrm{p}(\theta)$$

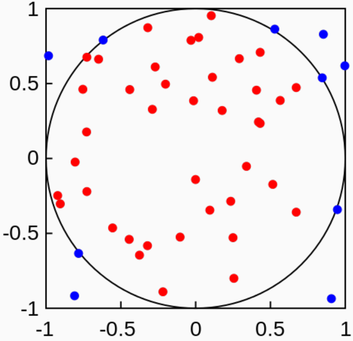
- Model specifies prior, likelihood, and data.
- The prior distribution reflects our knowledge of the data - use informative priors.
  - Prior Choice Recommendations
- We can compute the mean, median, mode, and std deviation of the posterior probability distribution $\mathrm{p}\,(\theta|\mathrm{y})$
- Quantiles of the posterior probability distribution provide *credible intervals*.

Use all of your knowledge - account for your uncertainty!

Monte Carlo Simulation: Calculate π

Computing $\pi = 3.14...$ via simulation is the textbook application of Monte Carlo methods.

- Generate points (x,y) uniformly at random within range (-1, 1)
- Calculate proportion within unit circle: $x^2 + y^2 < 1$
- Area of the square is 4
- Area of a circle is πr^2
- Area of the unit circle is π
- Ratio of points inside circle to total points is $\frac{\pi}{4}$
- $\pi =$ points inside circle × 4



Monte Carlo Simulation: Calculate π using Python

```
import numpy as np
def estimate_pi(n: int) -> float:
    xs = np.random.uniform(-1,1,n)
    ys = np.random.uniform(-1,1,n)
    dist_to_origin = [x**2 + y**2 for x,y in zip(xs, ys)]
    in_circle = sum(dist < 1 for dist in dist_to_origin)
    pi = float(4 * (in_circle / n))
    return pi
```

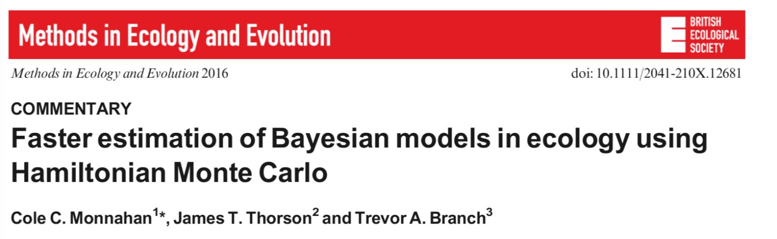
Sample size, estimate, elapsed time

N	Pi.estimate	elapsed.time
100	3.5	0.0008
10,000	3.15	0.03
1,000,000	3.139	3.2
100,000,000	3.1413	323.8

precision: $\dfrac{1}{\sqrt{N}}$

Stan's secret sauce: HMC-NUTS sampler

- **Hamiltonian Monte Carlo** - algorithm for efficient MCMC sampling.
- **NUTS sampler** - Hoffman and Gelman, 2014 - efficient convergence.
- No longer secret: PyMC3, Edward, Pyro use HMC-NUTS.
- A really nice overview:
  - Monnahan, 2016.



Methods in Ecology and Evolution

BRITISH ECOLOGICAL SOCIETY

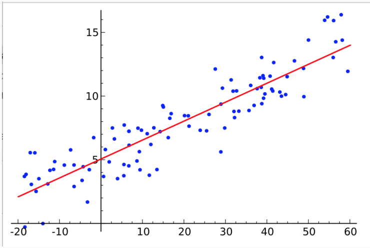
Methods in Ecology and Evolution 2016 doi: 10.1111/2041-210X.12681

COMMENTARY

Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo

Cole C. Monnahan[1]*, James T. Thorson[2] and Trevor A. Branch[3]

Review: linear regression



- Data consist of pairs x inputs, y outputs
- Regression model fits parameters for line: α intercept, β slope
- Regression formula: $\mathrm{y} = \alpha + \beta * \mathrm{x} + \epsilon$,
  where ϵ is random noise which has distribution $\epsilon \sim \mathcal{N}(0, \sigma)$
- Corresponding Stan program statement:

```
    y ~ normal(alpha + beta * x, sigma);
```

From Simple Linear Regression to Multi-level Models

Example: Major League Baseball Player Batting Ability

- Stan Case Study:
  "Hierarchical Partial Pooling for Repeated Binary Trials"
- Data: batting records for Major League Baseball players.
  - given number of hits in first 45 at-bats,
    estimate probability of a hit for a single at-bat.
- Three models:
  - estimate average player ability
  - estimate individual player abilities
  - estimate both average ("group-level") and individual ("random") abilities
- Visualization: compare model estimates

Hierarchical Partial Pooling for Repeated Binary Trials

Modelling Player Batting Ability

Bernoulli distribution - single trial (at-bat)

- Bernoulli distribution: If $\theta \in [0,1]$, then for $y \in \{0,1\}$,

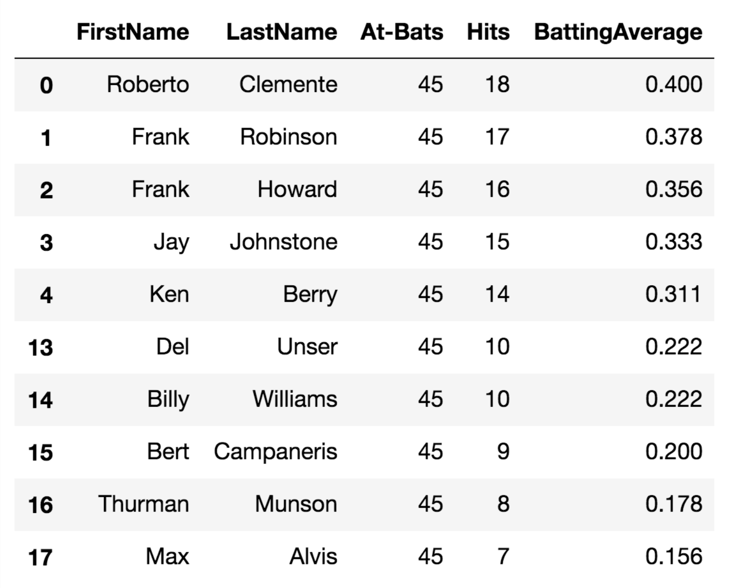
$$\text{Bernoulli}(y \mid \theta) = \begin{cases} \theta & \text{if } y = 1, \text{ and} \\ 1 - \theta & \text{if } y = 0. \end{cases}$$

Binomial distribution - repeated Bernoulli trials (n at-bats)

- Binomial distribution: Suppose $N \in \mathbb{N}$ and $\theta \in [0,1]$, and $n \in \{0, \ldots, N\}$.

$$\text{Binomial}(n \mid N, \theta) = \binom{N}{n} \theta^n (1-\theta)^{N-n}.$$

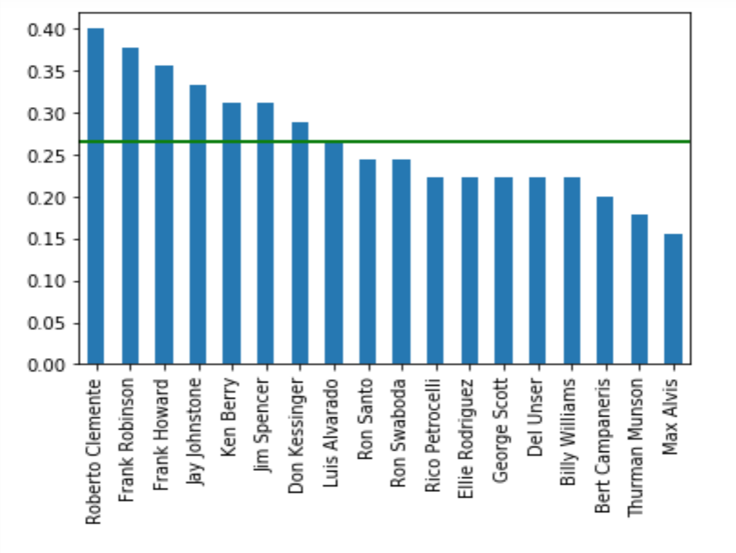
Data: First 45 At-Bats for MLB Players in 1975



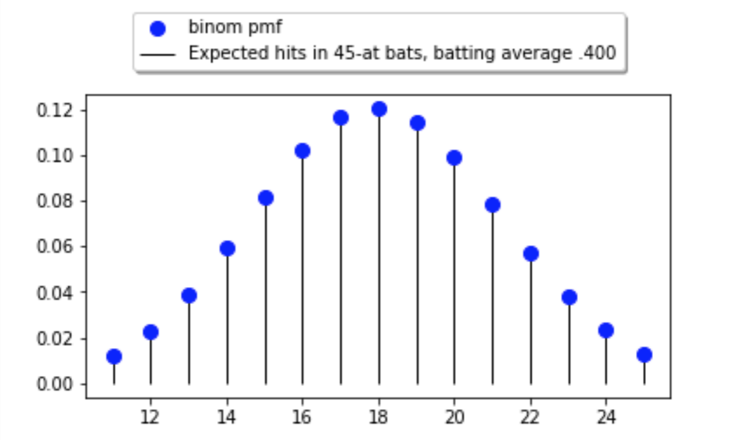
	FirstName	LastName	At-Bats	Hits	BattingAverage
0	Roberto	Clemente	45	18	0.400
1	Frank	Robinson	45	17	0.378
2	Frank	Howard	45	16	0.356
3	Jay	Johnstone	45	15	0.333
4	Ken	Berry	45	14	0.311
13	Del	Unser	45	10	0.222
14	Billy	Williams	45	10	0.222
15	Bert	Campaneris	45	9	0.200
16	Thurman	Munson	45	8	0.178
17	Max	Alvis	45	7	0.156

Preliminary data analysis

Plot individual batting averages and average batting average



Binomial Distribution: Roberto Clemente



Hierarchical Partial Pooling for Repeated Binary Trials

Same input data for all models

```
data {
  int<lower=0> N;            // items
  int<lower=0> K[N];         // initial trials
  int<lower=0> y[N];         // initial successes
}
```

Stan syntax similar to C/C++/Java:

- Variables are strongly typed.
- Blocks enclosed by curly braces.
- Top-level blocks are named (labeled), fixed ordering of named blocks.
- Statements in a block are executed in order.
- Semi-colons terminate statement.
- C++-style line-based (//) and bracketed comments (/* ... */).

Hierarchical Partial Pooling for Repeated Binary Trials

Model 1: Players are all alike - complete pooling

```
parameters {
  real<lower=0, upper=1> phi;  // chance of success
}
model {
  y ~ binomial(K, phi);  // likelihood
}
```

Model 2: All players are different - no pooling

```
parameters {
  vector<lower=0, upper=1>[N] theta; // chance of success
}
model {
  y ~ binomial(K, theta);  // likelihood
}
```

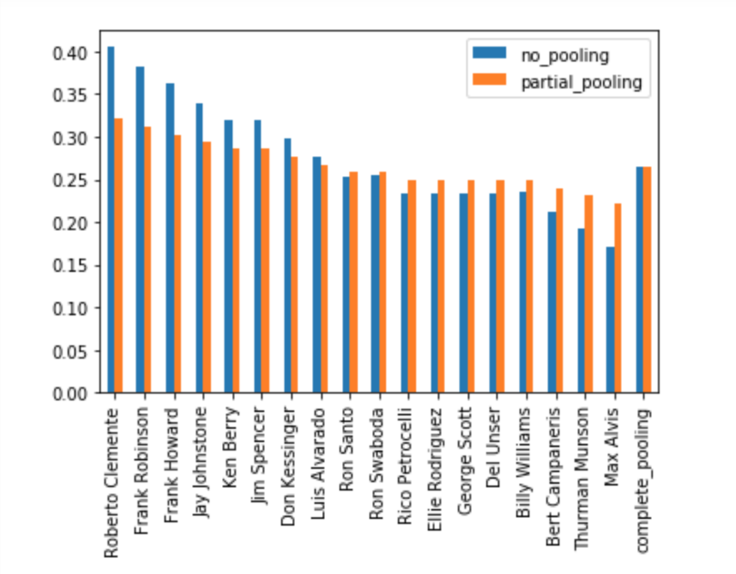
Hierarchical Partial Pooling for Repeated Binary Trials

Model 3: Multi-level Model!!

```
parameters {
  real<lower=0, upper=1> phi;   // population chance of success
  real<lower=1> kappa;          // population concentration
  vector<lower=0, upper=1>[N] theta;  // chance of success
}
model {
  kappa ~ pareto(1, 1.5);                        // hyperprior
  theta ~ beta(phi * kappa, (1 - phi) * kappa);  // prior
  y ~ binomial(K, theta);                        // likelihood
}
```

- Resulting model contains parameters `phi`, `kappa`, and `theta`
  - `phi`, `kappa` - general player population (“group effects”)
  - `theta` - individual player ability (“random effects”)

Model Comparison: Complete-, No-, and Partial Pooling



Another Kind of Ball Game

2019 FIFA Women's World Cup (WWC)

Model Comparison: Complete-, No-, and Partial Pooling

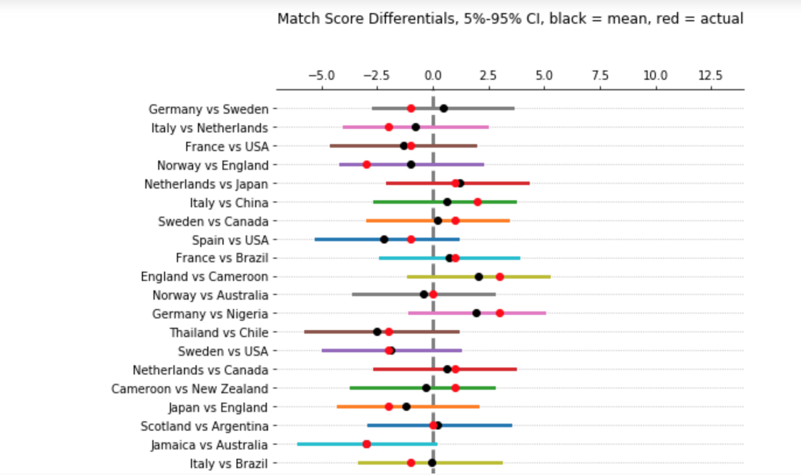
Let's do the iPython notebook

Ranking via Paired Comparisons - Player vs. Player

- Stan Case Study:
  "The Bradley-Terry Model of Ranking via Paired Comparisons"
- Data: a series of paired comparisons, e.g.:
  - a match between two players or teams
  - consumer preference between two items
- Estimate player/team ability based on match outcome.
- Ability is a linear model with group- and individual-level components.

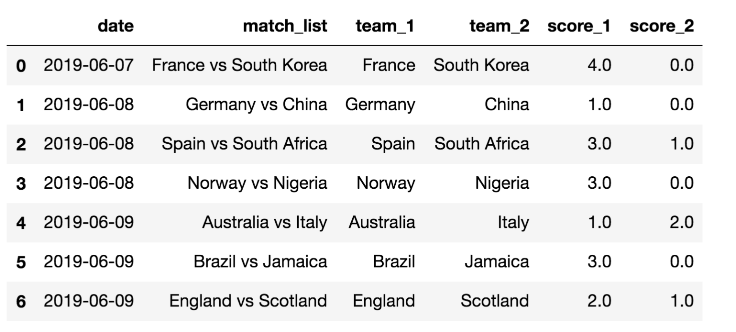
2019 FIFA Women's World Cup

Quantity of interest: difference in goals scored by each team



World Cup Model - Data

Outcomes of 48 matches - through quaterfinals



	date	match_list	team_1	team_2	score_1	score_2
0	2019-06-07	France vs South Korea	France	South Korea	4.0	0.0
1	2019-06-08	Germany vs China	Germany	China	1.0	0.0
2	2019-06-08	Spain vs South Africa	Spain	South Africa	3.0	1.0
3	2019-06-08	Norway vs Nigeria	Norway	Nigeria	3.0	0.0
4	2019-06-09	Australia vs Italy	Australia	Italy	1.0	2.0
5	2019-06-09	Brazil vs Jamaica	Brazil	Jamaica	3.0	0.0
6	2019-06-09	England vs Scotland	England	Scotland	2.0	1.0

World Cup Model - Data

Soccer Power Index

```
rank   country       spi
1          USA  97.20623
2       France  95.31747
3      Germany  94.17161
4    Australia  93.13200
5  Netherlands  92.70844
...
20    Jamaica  59.78145
21   Scotland  50.75001
22   Thailand  50.43348
23      Chile  48.96239
24  Argentina  37.08972
```

- from 538 by Nate Silver

World Cup Model - Data

```
data {
  int<lower=0> I;      // number of teams
  vector[I] spi;       // per-team ranking

  int<lower=0> N;      // number of matches
  int team_1[N];       // per-match data
  int team_2[N];
  vector[N] score_1;
  vector[N] score_2;
}
```

- Soccer Power Index (SPI) score:
    - per-team ability estimate (prior to WC play)
- Game outcomes:
    - team_1 id, team_2 id, team_1 score, team_2 score

World Cup Model - Transformed Data

```
transformed data {
  vector[N] y = score_1 - score_2;
  vector[I] spi_std;                // standardize SPI
  real spi_mean = mean(spi);
  real spi_sd = sd(spi);
  for (i in 1:I) {
    spi_std[i] = (spi[i] - spi_mean)/spi_sd;
  }
}
```

- The `transformed data` block
  - Executed only once, on model instantiation
  - Declare and define new data variables
  - Create random data using RNG functions
- Compute match outcome as ($\text{score}_1 - \text{score}_2$):
- Standardize SPI ratings - center on 0, scale variance to 1.

World Cup Model - Model

```
parameters {
  vector[I] alpha;         // per-team - "random effects"
  real beta;               // shared - "group effects"

  real<lower=0> sigma_a;   // scale of per-team variation
  real<lower=0> sigma_y;   // noise term in our estimate
}
transformed parameters {

  // model ability - will be included in sample output
  vector[I] ability = alpha * sigma_a + beta * spi_std;

}
model {
  y ~ normal(ability[team_1] - ability[team_2], sigma_y);

  // priors on all parameters
  alpha ~ normal(0, 1);        beta ~ normal(0, 2.5);
  sigma_a ~ normal(0, 2.5);    sigma_y ~ normal(0, 2.5);
}
```

World Cup Model - Generated Quantities

Compute *quantities of interest* in `generated quantities`

- Executed per iteration, once sampler has obtained a valid draw from the posterior.

Replicate observed data `y` as `y_rep`

```
generated quantities {
  // generate replicated data using estimated parameters
  vector[N] y_rep;
  for (n in 1:N) {
    y_rep[n] = normal_rng(ability[team_1[n]] - ability[team_2[n]],
                          sigma_y);
  }
}
```

Replicate statement uses RNG functions - cf. sampling statement

```
  y ~ normal(ability[team_1] - ability[team_2], sigma_y);
```

World Cup Model - Generated Quantities

Predict outcome of future matches using current estimate of team abilities

- data block - parallel arrays: team$_1$ id, team$_2$ id

```
int team_1_semis[2];  // these hold indices into
int team_2_semis[2];  // the vector of abilities
```

- `generated quantities` block

```
// predict the semi-finals
vector[2] semis;
for (n in 1:2) {
  semis[n] = normal(ability[team_1_semis[n]] - ability[team_2_semis[n]],
                    sigma_y);
}
```

2019 FIFA Women's World Cup

Let's do the iPython notebook

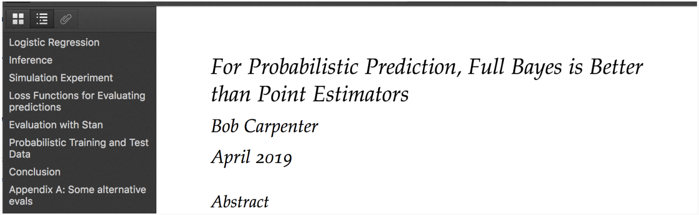
Evaluating Inference: *The Big Data Baking Show*

Given a model and dataset:

- subdivide into training, test data
- use training data to infer model parameters
- plug inferred params into data generating model
- use test data inputs (xs) to predict test data (ys)

May the best inference algorithm win!

Evaluating Inference



- Logistic Regression
- Inference
- Simulation Experiment
- Loss Functions for Evaluating predictions
- Evaluation with Stan
- Probabilistic Training and Test Data
- Conclusion
- Appendix A: Some alternative evals

For Probabilistic Prediction, Full Bayes is Better than Point Estimators

Bob Carpenter

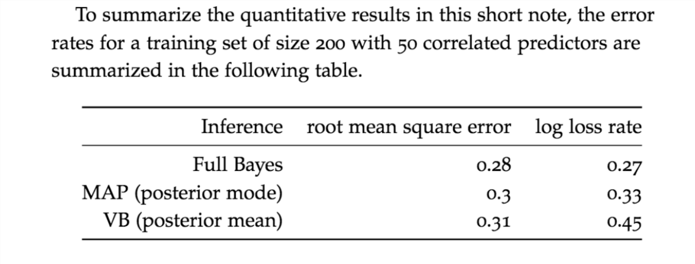
April 2019

Abstract

Stan Case Study: Probalistic Prediction

- The data generating model: logistic regression
- Use the model to simulate data - true values of parameters are known
- Fit the model using simulated data
- Evaluate predictions made by estimators using different inference algorithms

Master Baker: Full Bayes!



To summarize the quantitative results in this short note, the error rates for a training set of size 200 with 50 correlated predictors are summarized in the following table.

Inference	root mean square error	log loss rate
Full Bayes	0.28	0.27
MAP (posterior mode)	0.3	0.33
VB (posterior mean)	0.31	0.45

Pdf on GitHub

Evaluating Inference: Cooking with CmdStanPy

- Full Bayesian Inference
  - Stan's NUTS-HMC sampler generates a sample from the posterior distribution, compute statistics.
  - CmdStanPy `CmdStanModel` class method `sample`
- Maximum a posteriori approximate inference (MAP estimates)
  - Stan's optimization algorithms find the modes of the density specified by a Stan program ("penalized MLE").
  - CmdStanPy `CmdStanModel` class method `optimize`
- Variational approximation inference
  - Variational Bayes (VB) tries to find an approximate distribution matching the posterior and extracts the posterior mean values
  - CmdStanPy `CmdStanModel` class method `variational`

Questions Welcome! and Massive Thanks!

NYC PyLadies, especially:

- Nitya Mandyam
- Melissa Ferrari
- Felice Ho

NYC WiMLDS, especially:

- Reshama Shaikh

Team Stan!

References

- CmdStanPy
  - PyPi
  - Github
  - Online Documentation

- Stan Documentation
- Stan Forums
- Stan Case Studies:
  - Pooling with Hierarchical Models for Repeated Binary Trials
  - The Bradley-Terry Model of Ranking via Paired Comparisons

- Hoffman and Gelman, 2014: The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo
- Monnahan, 2016: Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo.