Pyro

Bayesian Data Analysis with PPLs: Bayesian Regression in Pyro

UBER AI Labs

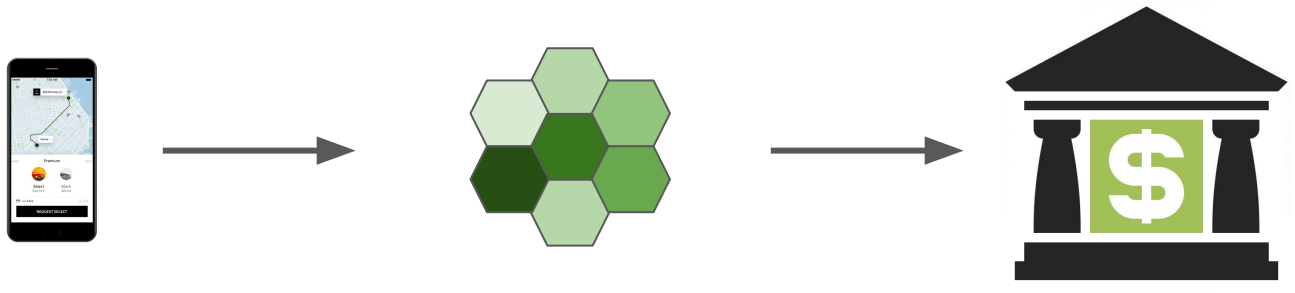
In this tutorial:

1. Why probabilistic machine learning?
2. Building a probabilistic regression model in Pyro
3. Approximate inference in Bayesian regression with SVI and MCMC
4. Pros and cons of SVI vs MCMC: subsampling, bias

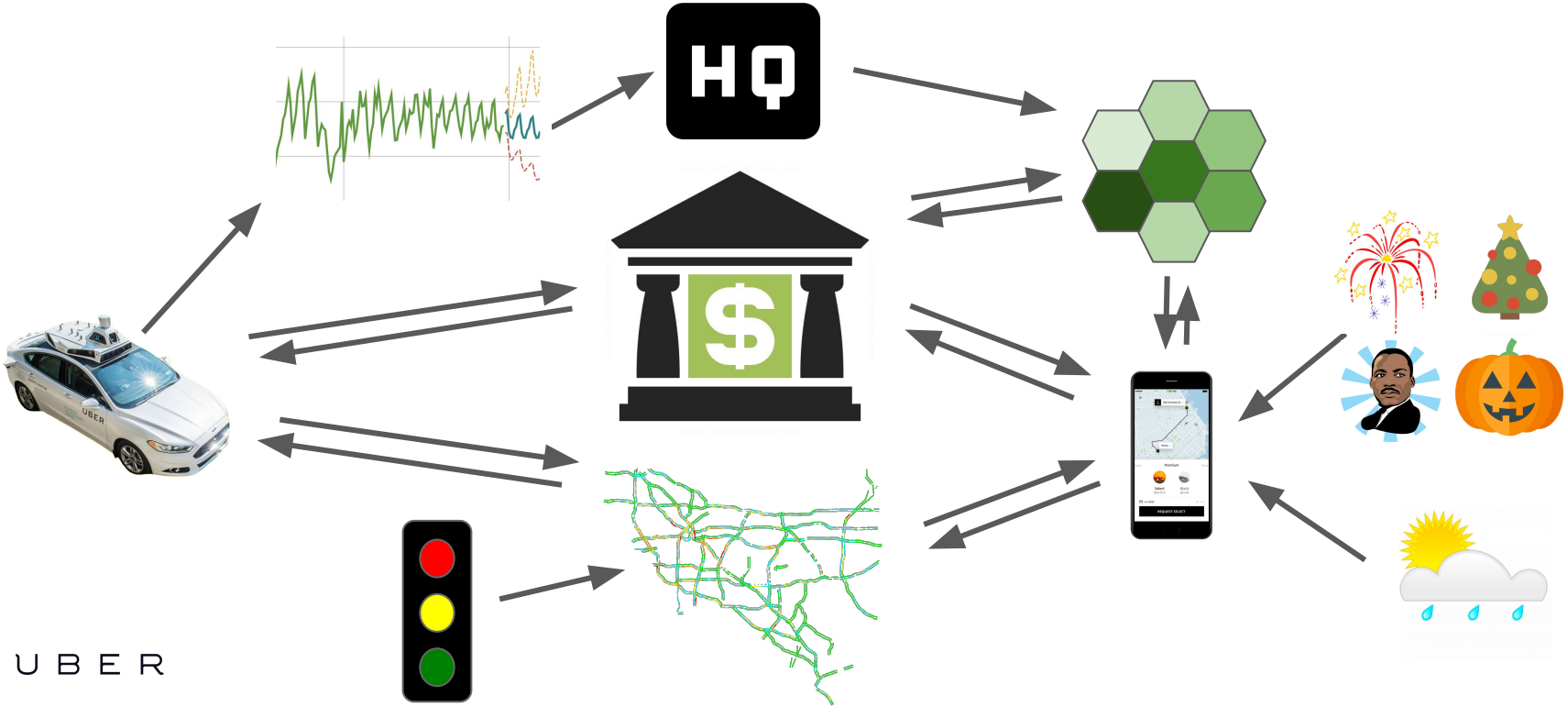
Based on the following Pyro tutorials:

- Bayesian regression 1: here
- Bayesian regression 2: http://pyro.ai/examples/bayesian_regression.html

Machine learning problems in theory

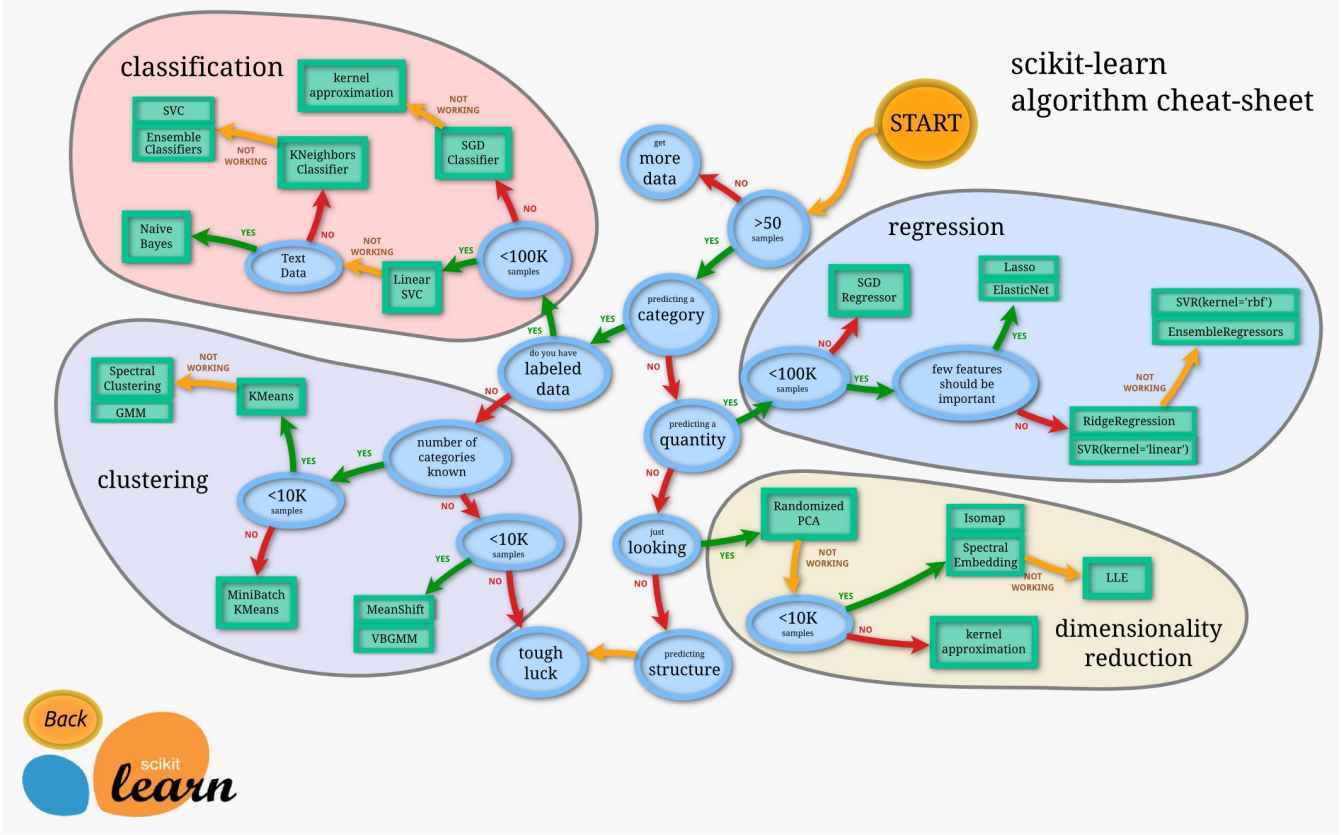


Machine learning problems in practice



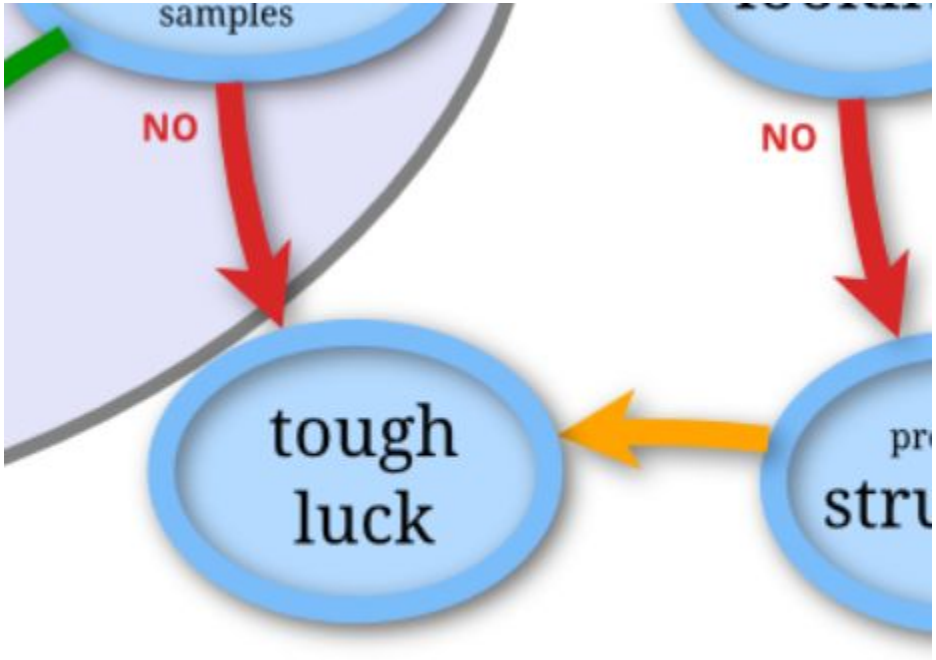
UBER

Machine learning solutions in practice



http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Machine learning solutions in practice



Probabilistic Modeling and Inference

Unified account of generalization, regularization, unsupervised learning, calibrated uncertainty estimates, multitask/transfer learning, model composition, ...

Everything follows from two simple rules:

Sum rule: $P(x) = \sum_y P(x, y)$

Product rule: $P(x, y) = P(x)P(y|x)$

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}$$

$P(\mathcal{D}\|\theta)$	likelihood of θ
$P(\theta)$	prior probability of θ
$P(\theta\|\mathcal{D})$	posterior of θ given $\mathcal{D}$

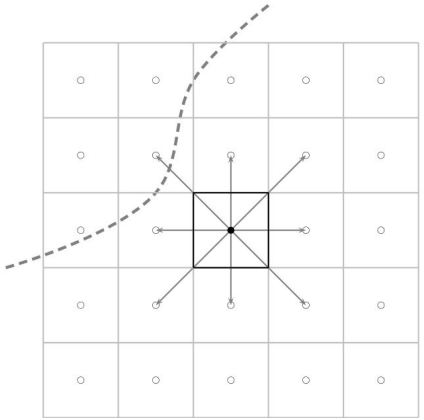
(h/t Zoubin)

Predicting GDP from terrain

Q: Did rugged terrain shield some African countries from the negative long-term economic effects of the slave trade?

We will attempt to estimate GDP in 2000 for African and non-African countries from a measure of terrain ruggedness

These predictions will necessarily have uncertainty, so we will use Bayesian linear regression



UBER

McElreath, D., *Statistical Rethinking, Chapter 7*, 2016

Nunn, N. & Puga, D., *Ruggedness: The blessing of bad geography in Africa*

Probabilistic regression: model

We model expected log(GDP) as a linear function of terrain ruggedness and whether the country is in Africa or not:

```
def model(is_cont_africa, ruggedness):
    ...
    mu = a + b_a * is_cont_africa + b_r * ruggedness + \
        b_ar * is_cont_africa * ruggedness
    ...
```

Probabilistic regression: model

We model the data as drawn from a `Normal` around the predicted mean:

```
def model(is_cont_africa, ruggedness):
    ...
    mu = a + b_a * is_cont_africa + b_r * ruggedness + \
        b_ar * is_cont_africa * ruggedness
    return pyro.sample("obs", Normal(mu, sigma))
```

Probabilistic regression: model

We put simple, independent priors on all regression coefficients:

```
def model(is_cont_africa, ruggedness):
    a = pyro.sample("a", Normal(8., 1000.))
    b_a = pyro.sample("bA", Normal(0., 1.))
    b_r = pyro.sample("bR", Normal(0., 1.))
    b_ar = pyro.sample("bAR", Normal(0., 1.))
    sigma = pyro.sample("sigma", Uniform(0., 10.))

    mu = a + b_a * is_cont_africa + b_r * ruggedness + \
        b_ar * is_cont_africa * ruggedness

    ...
```

Probabilistic regression: “mean-field” guide

An especially simple guide: an independent `Normal` for each latent variable:

```
def guide(is_cont_africa, ruggedness, data):
    ...
    loc_a = pyro.param("loc_a", ...)
    scale_a = pyro.param("scale_a", ...)
    a = pyro.sample("a", Normal(loc_a, scale_a))
    ...
    sigma_dist = Normal(...)
    sigma = pyro.sample("sigma", sigma_dist)
```

Inference as optimization: understanding the ELBO

We want a loss function that is always non-negative and is zero when our guide q(**z**) is equal to the true posterior p(**z** | x):

$$\mathcal{L}(q_\phi(\mathbf{z}), p(\mathbf{z}|x)) \geq 0 \qquad \mathcal{L}(p(\mathbf{z}|x), p(\mathbf{z}|x)) = 0$$

One such loss function is the KL divergence:

$$\mathrm{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}|x)) = \mathbb{E}_q[\log q_\phi(\mathbf{z}) - \log p(\mathbf{z}|x)]$$

Inference as optimization: understanding the ELBO

Evaluating the KL divergence requires knowing the true posterior p(**z** | x). Instead we use an approximation that only requires evaluating p(x, **z**):

$$\begin{aligned}\mathrm{ELBO} &= \mathbb{E}_q[\log p(x, \mathbf{z}) - \log q_\phi(\mathbf{z})] \\ &= C - \mathrm{KL}(q_\phi(\mathbf{z}) || p(\mathbf{z}|x))\end{aligned}$$

Because the constant C does not depend on the guide, maximizing the ELBO minimizes the KL between guide q(**z**) and true posterior p(**z** | x)

Inference as optimization: training the guide with SVI

```
svi = SVI(conditioned_model,
          guide,
          Adam({"lr": .005}),
          loss=Trace_ELBO())


...
for i in range(10000):
    svi.step(is_cont_africa, ruggedness, log_gdp)
```

MCMC in Pyro: Hamiltonian Monte Carlo (HMC)

Pyro provides a No U-Turn Sampler MCMC kernel (as in Stan, PyMC3) for scalable, asymptotically unbiased inference:

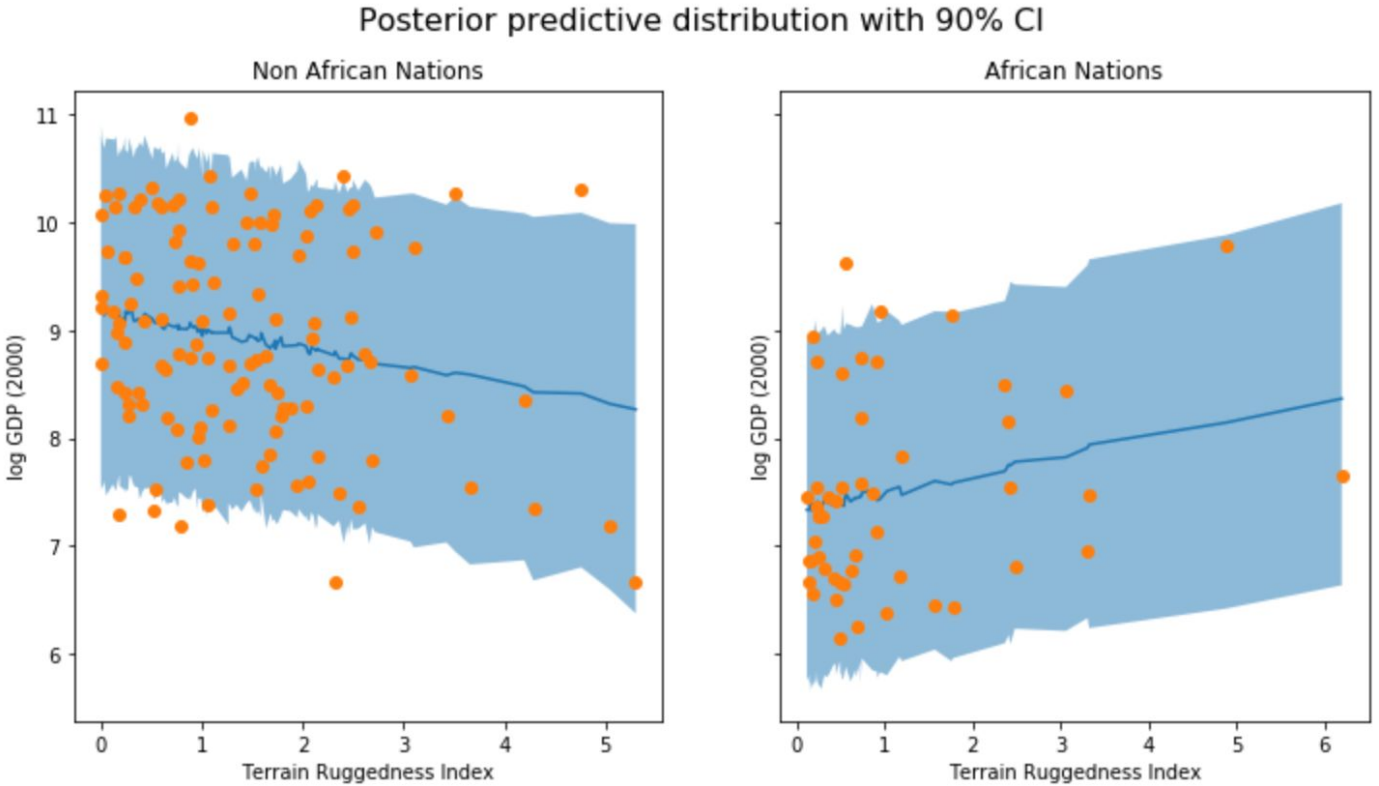
```
nuts_kernel = pyro.infer.NUTS(conditioned_model, adapt_step_size=True)
```

We apply the kernel and save the results with `pyro.infer.MCMC`

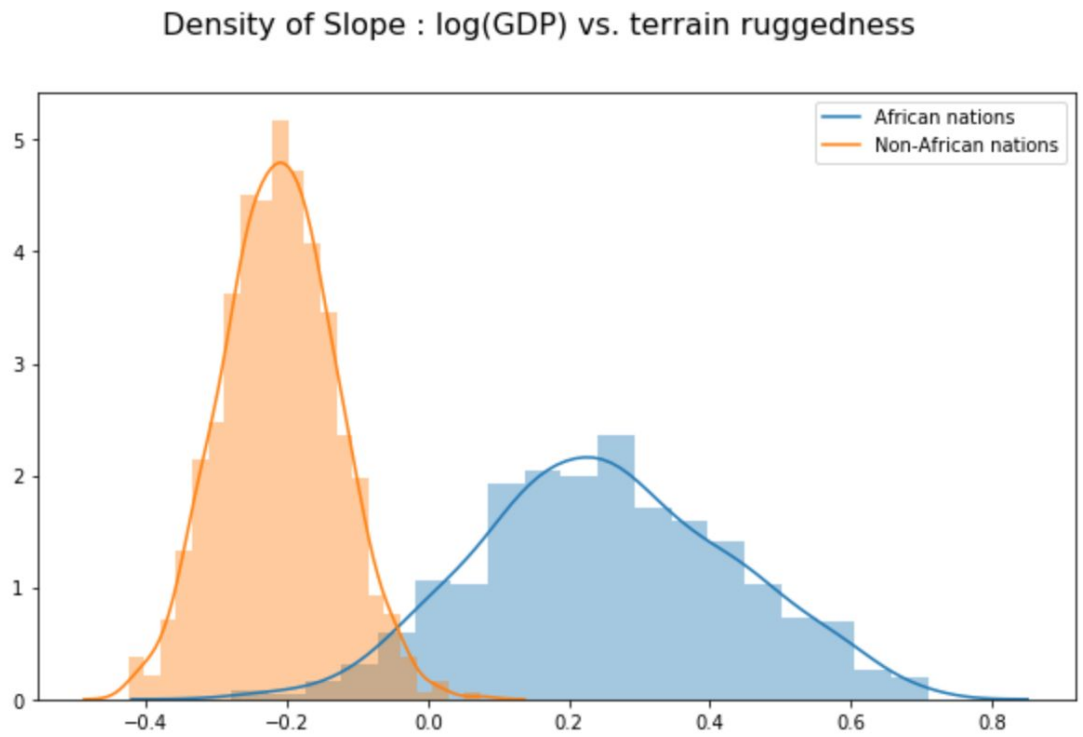
```
hmc_posterior = pyro.infer.MCMC(nuts_kernel,
                                num_samples=1000,
                                warmup_steps=200)

hmc_posterior = hmc_posterior.run(is_cont_africa, ruggedness, log_gdp)
```

Bayesian regression: results with HMC



Bayesian regression: results with HMC



UBER

SVI vs MCMC: subsampling

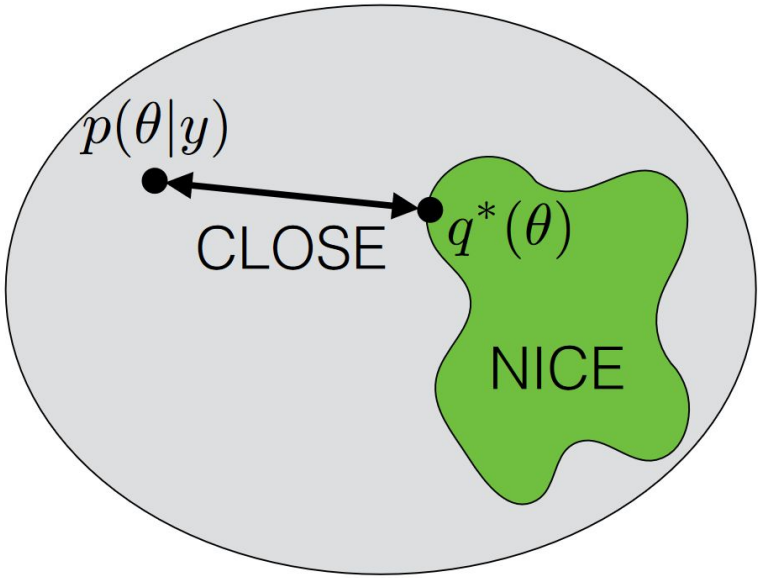
In SVI, we can subsample latent and observed variables with `pyro.iarange` which randomly samples a set of indices and rescales probabilities:

```
def subsample_model(is_cont_africa, ruggedness, data):
    a = pyro.sample("a", dist.Normal(8., 1000.))
    ...
    with pyro.iarange("data", len(ruggedness), subsample_size=1) as ix:
        pyro.sample("obs", dist.Normal(mu, sigma), obs=data[ix])
```

With subsampling, we can apply SVI to much larger models and data

SVI vs MCMC: biased approximation

Our parameterized family of guides may not include the true posterior:

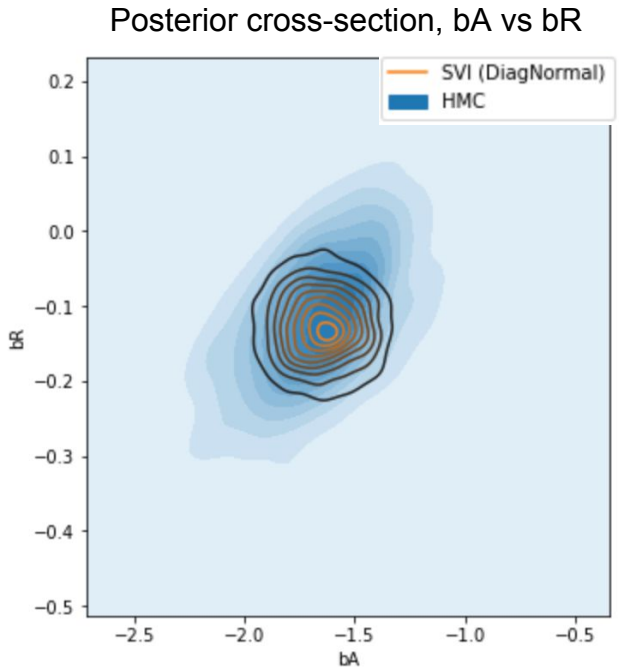


UBER

Variational Bayes and Beyond: Bayesian Inference for Big Data, Broderick, T.

SVI vs MCMC: biased approximation and underdispersion

Variational inference with simple guides underestimates posterior uncertainty...

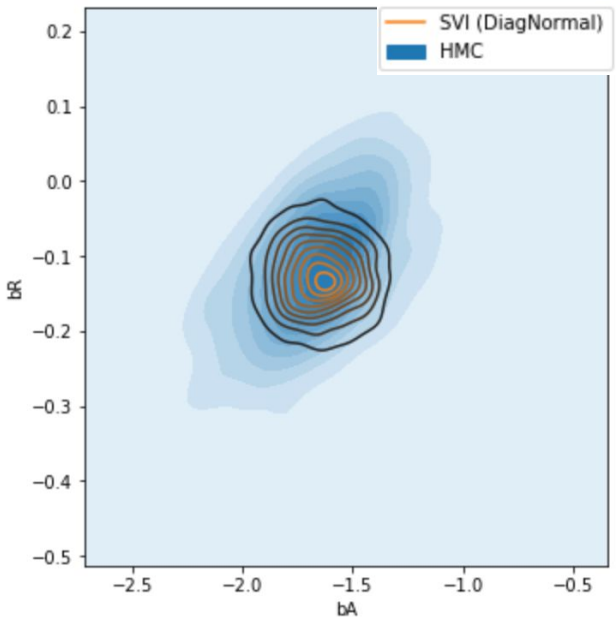


UBER

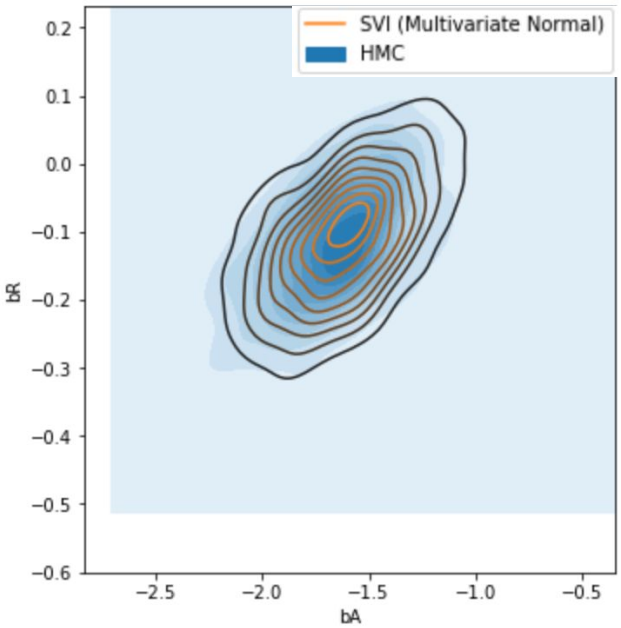
SVI vs MCMC: mitigating underdispersion

...but more expressive guides may do a better job of reducing this bias:

Posterior cross-section, bA vs bR



Posterior cross-section, bA vs bR



UBER

Recap

1. A case for probabilistic machine learning
2. Built a probabilistic regression model in Pyro
3. Approximate inference in Bayesian regression with SVI and MCMC
4. Pros and cons of SVI vs MCMC: subsampling, bias

Coming up: using Pyro to build and learn powerful black-box generative models based on deep neural networks

pyro.ai

Eli Bingham

JP Chen

Martin Jankowiak

Theo Karaletsos

Fritz Obermeyer

Neeraj Pradhan

Rohit Singh

Paul Szerlip

Noah Goodman

Special thanks to

Paul Horsfall
Dustin Tran
Soumith Chintala
Adam Paszke
Du Phan

Would you like to know more?

Pyro tutorials web page: http://pyro.ai/examples/index.html

Another walkthrough of Bayesian regression with SVI using advanced Pyro features:

http://pyro.ai/examples/bayesian_regression.html

A more complicated regression example demonstrating both SVI and MCMC:

https://github.com/uber/pyro/tree/dev/examples/eight_schools

Pyro MCMC, HMC, and NUTS documentation:

http://docs.pyro.ai/en/0.2.1-release/mcmc.html

Constraining parameters and guides in SVI

In SVI, we must ensure that all ELBO terms are finite:

$$\mathrm{ELBO} \equiv \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right]$$

Constraining parameter values in SVI

Parameters of distributions must be in the right range, assisted by `torch.distributions.constraints`

```
def guide(is_cont_africa, ruggedness, data):
    ...
    loc_a = pyro.param("loc_a")
    scale_a = pyro.param("scale_a", constraint=constraints.positive)
    a = pyro.sample("a", Normal(loc_a, scale_a))
    ...
```

Matching random variable supports in SVI

Distribution pairs in models and guides must have the same support, assisted by `torch.distributions.biject_to` and `TransformedDistribution`

```
def guide(is_cont_africa, ruggedness, data):
    ...
    sigma_transform = torch.distributions.biject_to(constraints.positive)
    sigma_dist = TransformedDistribution(Normal(...), sigma_transform)
    sigma = pyro.sample("sigma", sigma_dist)
```