Workshop: Intro to Bayesian Learning
Lecture 6 - Implementing Bayesian Learning with Probabilistic Programming

Mattias Villani

Department of Statistics
Stockholm University

mattiasvillani.com @matvil @matvil mattiasvillani

Overview

- **Probabilistic programming**
- **Turing**
- **Stan**

Probabilistic programming languages for Bayes

- **Stan** is a probabilistic programming language for Bayes based on HMC.
- C++ using the R package `rstan`. Bindings from Python.

- **Turing.jl** is a probabilistic programming language in Julia.
- Written in `Julia`, which is fast natively.

HMC sampling for iid normal model in rstan

```r
library(rstan)

# Define the Stan model
stanModelNormal = '
// The input data is a vector y of length N.
data {
  // data
  int<lower=0> N;
  vector[N] y;
  // prior
  real mu0;
  real<lower=0> kappa0;
  real<lower=0> nu0;
  real<lower=0> sigma20;
}

// The parameters in the model
parameters {
  real theta;
  real<lower=0> sigma2;
}

model {
  sigma2 ~ scaled_inv_chi_square(nu0, sqrt(sigma20));
  theta ~ normal(mu0,sqrt(sigma2/kappa0));
  y ~ normal(theta, sqrt(sigma2));
}
'

# Set up the observed data
data <- list(N = 5, y = c(15.77, 20.5, 8.26, 14.37, 21.09))

# Set up the prior
prior <- list(mu0 = 20, kappa0 = 1, nu0 = 5, sigma20 = 5^2)

# Sample from posterior using HMC
fit <- stan(model_code = stanModelNormal, data = c(data,prior), iter = 10000 )
```

HMC sampling for iid normal model in Turing.jl

```julia
using Turing

ScaledInverseChiSq(ν,τ²) = InverseGamma(ν/2,ν*τ²/2) # Scaled Inv-χ² distribution

# Setting up the Turing model:
@model function iidnormal(x, μ₀, κ₀, ν₀, σ²₀)
    σ² ~ ScaledInverseChiSq(ν₀, σ²₀)
    θ ~ Normal(μ₀, √(σ²/κ₀))  # prior
    n = length(x)  # number of observations
    for i in 1:n
        x[i] ~ Normal(θ, √σ²) # model
    end
end

# Set up the observed data
x = [15.77,20.5,8.26,14.37,21.09]

# Set up the prior
μ₀ = 20; κ₀ = 1; ν₀ = 5; σ²₀ = 5^2

# Settings of the Hamiltonian Monte Carlo (HMC) sampler.
α = 0.8
postdraws = sample(iidnormal(x, μ₀, κ₀, ν₀, σ²₀), NUTS(α), 10000, discard_initial = 1000)
```

Modeling the number of bidders in eBay auctions

variable	description	data type	original range
nbids	number of bids	counts	$[0, 12]$
bookvalue	coin's book value	continuous	$[7.5, 399.5]$
startprice	seller's reservation price / book value	continuous	$[0, 1.702]$
minblemish	minor blemish	binary	$[0, 1]$
majblemish	major blemish	binary	$[0, 1]$
negfeedback	large negative feedback score	binary	$[0, 1]$
powerseller	large quantity seller	binary	$[0, 1]$
verified	verified seller on ebay	binary	$[0, 1]$
sealed	unopened package	binary	$[0, 1]$

- **Poisson regression**

$$y_i | \boldsymbol{x}_i \sim \mathrm{Poisson}(\lambda_i)$$

$$\lambda_i = \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})$$

HMC sampling for Poisson regression in Turing.jl

```
using Turing

# Setting up the poisson regression model
@model function poissonReg(y, X, τ)
    p = size(X,2)
    β ~ filldist(Normal(0, τ), p)  # all βⱼ are iid Normal(0, τ)
    λ = exp.(X*β)
    n = length(y)
    for i in 1:n
        y[i] ~ Poisson(λ[i])
    end
end

# HMC sampling from posterior of β
τ = 10    # Prior standard deviation
α = 0.70  # target acceptance probability in NUTS sampler
model = poissonReg(y, X, τ)
chain = sample(model, Turing.NUTS(α), 10000, discard_initial = 1000)
```

- Poisson regression in rstan.

… or TuringGLM.jl with R's formula syntax

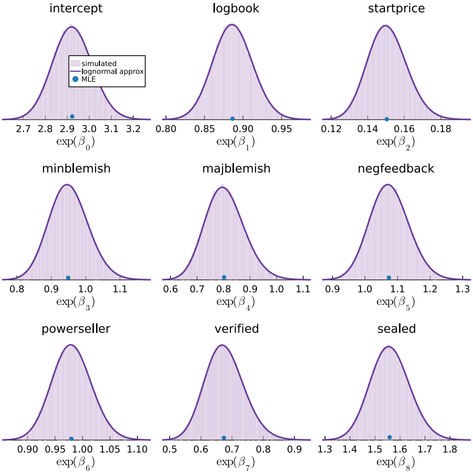
```julia
# Using TuringGLM.jl
using TuringGLM
fm = @formula(nbids ~ logbook + startprice + minblemish +
    majblemish + negfeedback + powerseller + verified + sealed)
model = turing_model(fm, ebay_df; model = Poisson)
chain = sample(model, NUTS(), 10000)
```

- Inspired by the brms package in R.

Marginal posteriors

- Multiplicative model

$$E(y|\boldsymbol{x}) = \exp(\beta_0+\beta_1 x_1+\beta_2 x_2) = \exp(\beta_0)\exp(\beta_1)^{x_1}\exp(\beta_2)^{x_2}$$



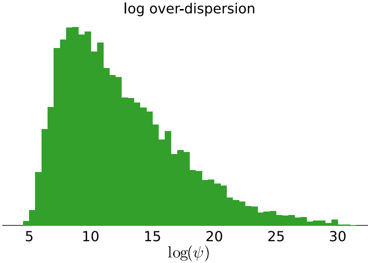
Negative binomial regression in Turing.jl

- **Negative binomial regression**

$$y_i | \boldsymbol{x}_i \sim \mathrm{NegBinomial}\left(\psi, p = \frac{\psi}{\psi + \lambda_i}\right), \quad \lambda_i = \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})$$

- Mean is still λ_i, but variance is larger: $Var(y_i) = \lambda_i(1 + \lambda_i/\psi)$.
- As $\psi \to \infty$ we get Poisson again.

```
# Negative binomial regression
@model function negbinomialReg(y, X, τ, μ₀, σ₀)
    p = size(X,2)
    β ~ filldist(Normal(0, τ), p)
    λ = exp.(X*β)
    ψ ~ LogNormal(μ₀, σ₀)
    n = length(y)
    for i in 1:n
        y[i] ~ NegativeBinomial(ψ, ψ/(ψ + λ[i]))
    end
end
```



Regression with horseshoe in Turing.jl

- **Horseshoe** prior

$$\beta_j|\lambda_j^2,\tau^2 \overset{\text{ind}}{\sim} N\left(0,\sigma^2\tau^2\lambda_j^2\right) \qquad \lambda_j \sim C^+(0,1) \qquad \tau \sim C^+(0,1)$$

```
# Define the half-Cauchy distribution
halfCauchy = truncated(Cauchy(0, 1); lower=0)

@model function BayesLinRegHS_turing(y, X, v₀, σ²₀, stdβ₀ = 100)
    p = size(X, 2) # X should not include intercept
    τ ~ halfCauchy
    λ ~ filldist(halfCauchy, p)
    σ² ~ ScaledInverseChiSq(v₀,σ²₀)
    β₀ ~ Normal(0, stdβ₀)
    β ~ MvNormal(zeros(p), Diagonal((λ .* τ).^2 .* σ²))
    y ~ MvNormal(β₀ .+ X*β, σ²*I)
end

# Prior hyperparameters
v₀ = 0.01; σ²₀ = 1.0

# Run HMC to sample from the posterior
@time hmc_draws = sample(BayesLinRegHS_turing(y, X[:,2:end], v₀, σ²₀),
    NUTS(), nDraws, n_adapts = 1000, n_chains = 1);

# Plot results
histogram(hmc_draws[:τ], bins = 100)
```

Regression with horseshoe using Gibbs sampling

```
for i ∈ 1:nSim

    # Compute things needed for sampling β,σ | λ, τ, y, X
    invΛ = diagm(1 ./ (λ.^2))
    Ω₀ = [1/(stdβ₀^2) zeros(1,p); zeros(p,1) (1/τ^2)*invΛ]
    Ωₙ = Symmetric(XX + Ω₀)
    invΩₙ = inv(Ωₙ)
    μₙ = Ωₙ\(XX*βhat)
    σ²ₙ = (ν₀*σ²₀ + (y-X*βhat)'*(y-X*βhat) + (μₙ-βhat)'*XX*(μₙ-βhat) + μₙ'*Ω₀*μₙ)/νₙ

    # Simulate from p(σ²|ψ²,y,X)
    σ² = rand(ScaledInverseChiSq(νₙ, σ²ₙ))
    σ²sim[i] = σ²
    σ = sqrt(σ²)

    # Simulate from p(β|ψ², σ²,y,X)
    β = rand(MvNormal(μₙ,σ²*invΩₙ))
    βsim[i,:] = β'

    # Simulate from p(λ | τ, β, σ², y, X)
    ν = rand.( ScaledInverseChiSq.(2, 1 .+ 1 ./ (λ.^2)) )
    λ = sqrt.(rand.(
        ScaledInverseChiSq.(2, 1 ./ ν .+ 0.5*(β[intercept+1:end]/(σ*τ)).^2 )
    ))

    # Simulate from p(τ | λ, β, σ², y, X)
    if estimate_τ
        ξ = rand( ScaledInverseChiSq.(2, 1 + 1/(τ^2)) )
        τ = sqrt(rand(
            ScaledInverseChiSq(p+1, (2/ξ + sum((β[intercept+1:end] ./ (σ*λ)).^2) )/(p+1) )
        ))
    end
    τsim[i] = τ

end
```