Turing.jl: Probabilistic programming with discrete random probability measures

Martin Trapp[1], Emile Mathieu[2], Maria Lomeli[3] and Hong Ge[4]

[1] Graz University of Technology, Graz, Austria
[2] University of Oxford, Oxford, UK
[3] Babylon Health, London, UK
[4] University of Cambridge, Cambridge, UK

Probabilistic programming with Turing.jl

Probabilistic modelling is a core component of a scientists' toolbox for incorporating uncertainties about the model parameters and noise in the data. Statistical models with a Bayesian nonparametric component are difficult to handle due to the infinite dimensionality which prevents the straightforward use of standard inference methods. Probabilistic programming languages -- such as Turing.jl [1] -- enable the **rapid development of new probabilistic models** while simultaneously automating statistical inference. This is made possible by separating the model definition from the inference scheme and by using generic inference algorithms.

In this work we present the **integration of Bayesian nonparametric priors into Turing.jl** which allows non-experts to use a variety of Bayesian nonparametric mixture models using a generic modelling framework. Turing.jl has an intuitive syntax and makes full use of the numerical capabilities in the Julia programming language, including all implemented probability distributions, and automatic differentiation.

[1] Ge, H., Xu, K., & Ghahramani, Z. (2018). Turing: a language for flexible probabilistic inference. In International Conference on Artificial Intelligence and Statistics (pp. 1682-1690).
[2] Bloem-Reddy, B., Mathieu, E., Foster, A., Rainforth, T., Ge, H., Lomeli, M., Ghahramani, Z., and Teh, Y.W. (2017), Sampling and inference for discrete random probability measures in probabilistic programs. Approximate Inference workshop at NIPS.

Turing.jl Syntax

Defines a Julia function that takes the observations as input.

The **@model** macro translates a Julia function into a Turing model.

Turing.jl recognizes **σ** to be a model parameter which is drawn from an inverse Gamma distribution.

Turing.jl interprets **x[i]** to be an observation distributed according to a Normal distribution.

To instantiate the probabilistic model we call the **gdemo(x)** function with a set of observations.

Turing.jl provides various inference algorithms (HMC, particle-based, ...) which can be applied to the instantiate model using the sample function.

```
using Turing

@model gdemo(x) = begin
    # Assumptions
    σ ~ InverseGamma(2, 3)
    μ ~ Normal(0, sqrt(σ))

    # Observations
    for i = 1:length(x)
        x[i] ~ Normal(μ, sqrt(σ))
    end
end

data = [0.157, -1.985, ..., 5.163]
model = gdemo(data)

sampler = HMC(10^5, 0.1, 5)
chn = sample(model, sampler)
describe(chn)
```

Non-parametric modelling using Turing.jl

Turing.jl admits **rapid development of new BNP priors** for parametric and non-parametric probabilistic modelling. Non-parametric priors are implemented by separating the representation from the random measure. Thus, providing a flexible interface for BNP modelling. The following example illustrates the implementation of the stick-breaking construction [4] of a Dirichlet process [5].

```
struct StickBreakingProcess <: Distribution
    rpm # random probability measure
end

minimum(d::StickBreakingProcess) = 0.0
maximum(d::StickBreakingProcess) = 1.0

struct DirichletProcess
    α # Concentration parameter
end

function rand(d::StickBreakingProcess{DirichletProcess})
    return rand(Beta(1, d.rpm.α))
end

function logpdf(d::StickBreakingProcess{DirichletProcess}, x)
    return logpdf(Beta(1, d.rpm.α), x)
end
```

Turing.jl currently provides implementations for well-known random measures and provides several constructions [2-4] for each.
In addition to existing inference algorithms, e.g. HMC, NUTS, MH, particle MCMC, inference in Turing is **compositional** and **easily hackable** making it possible to rapidly implement novel inference algorithms.

[3] Aldous, D. J. (1985). Exchangeability and related topics. In École d'Été de Probabilités de Saint-Flour XIII—1983 (pp. 1-198). Springer, Berlin, Heidelberg.
[4] Ishwaran, H., & James, L. F. (2001). Gibbs sampling methods for stick-breaking priors. Journal of the American Statistical Association, 96(453), 161-173.
[5] Blackwell, D., & MacQueen, J. B. (1973). Ferguson distributions via Pólya urn schemes. The annals of statistics, 1(2), 353-355.
[6] Mansinghka, V., Shafto, P., Jonas, E., Petschulat, C., Gasner, M., & Tenenbaum, J. B. (2016). Crosscat: A fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. The Journal of Machine Learning Research, 17(1), 4760-4808.

Example BNP Model (CrossCat [6] Variation)

$$y_p \sim \mathrm{PYCRP}(\{y_i \,|\, i \neq p\}, d, \theta) \qquad \forall p \in \{1, \ldots, P\}$$

$$z_n^v \sim \mathrm{CRP}(\{z_i^v \,|\, i \neq n\}, \alpha) \qquad \forall v \in \vec{y},\ \forall n \in \{1, \ldots, N\}$$

$$\mu_c^p \sim p(\mu \,|\, \mu_0) \qquad \forall v \in \vec{y},\ c \in \vec{z}^v \text{ and } p \text{ s.t. } y_p = v$$

$$x_{n,p} \sim p(x_{n,p} \,|\, \mu_c^p) \qquad \forall v \in \vec{y},\ c \in \vec{z}^v \text{ and } n \text{ s.t. } z_n^{y_p} = c$$

PYCRP

CRP

$p(\mu \,|\, \mu_0)$

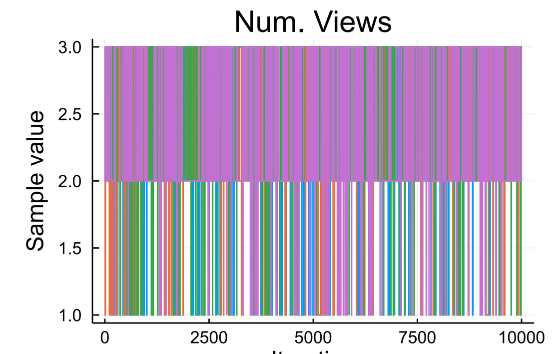
$p(x_{n,d} \,|\, \mu_c^p)$

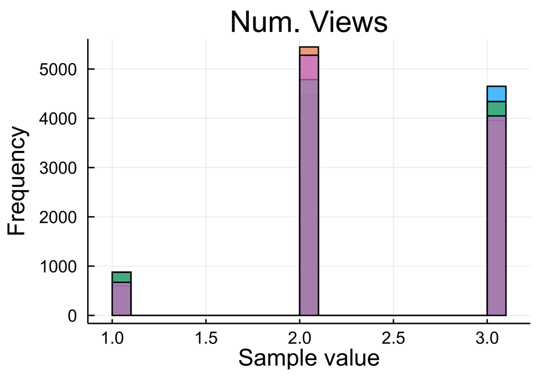
```
@model CrossCat(x, μ0, α, d, θ) = begin
  N,P = size(x)
  # assignments to views (dimensions)
  y = tzeros(Int, P)
  for p = 1:P
    pk = Int[sum(y .== k) for k = 1:maximum(y)]
    y[p] ~ ChineseRestaurantProcess(PitmanYorProcess(d, θ, p-1), pk)
  end
  μ = Vector{Vector{Vector{Real}}}(undef, maximum(y))
  # assignments of observations to clusters in each view
  z = TArray(TArray, maximum(y))
  for k = 1:maximum(y)
    μ[k], z[k] = Vector{Vector{Real}}(), tzeros(Int, N)

    # for each observation
    for n = 1:N
      # draw assignment to a cluster inside a view
      J = maximum(z[k])
      nj = Int[sum(z[k] .== j) for j = 1:J]
      z[k][n] ~ ChineseRestaurantProcess(DirichletProcess(α), nj)

      # make new cluster if necessary
      if z[k][n] > J
        push!(μ[k], Vector{Float64}(undef, P))
        μ[k][z[k][n]] ~ MvNormal(μ0, ones(P))
      end
      for p in findall(y .== k)
        x[n, p] ~ Normal(μ[k][z[k][n]][p])
      end
    end
end end
```

The Turing ecosystem provides multiple convergence diagnostics and visualisations to analyse sampling results.





Visit *https://turing.ml* for tutorials, API and further details.
BNP model zoo can be found under: http://tiny.cc/BNP12